MASTER IN COMPUTER SCIENCE

Extensions to a query system dedicated to a specification database

Geubelle, Bernard

Award date:
1983

Awarding institution:
University of Namur

FACULTÉS
UNIVERSITAIRES
N.D. DE LA PAIX
NAMUR

INSTITUT D'INFORMATIQUE

EXTENSIONS TO A QUERY SYSTEM

DEDICATED TO

A SPECIFICATION DATABASE

BERNARD GEUBELLE

Thesis presented in order to obtain
the master degree in Computer Science

Academic year 1982-1983

Adresse : rue Grandgagnage, 21, B-5000 NAMUR (Belgium) - ☎ 081 - 22 90 65 - Télex 59.222 FAC NAM-B

I would like to thank professor BODART who accepted to conduct this thesis. His criticisms and advices have highly contributed to the realization of this thesis.

I am also thankful to professor TEICHROEW and his team for his welcome in the ISDOS project of the University of Michigan. I am deeply grateful for the constant help of professor KANG during my stay in Ann Arbor.

Finally, I am also indebted to all the team of the ISDOS project in Namur whose advices were valuable to solve the technical problems encountered during the realization of this thesis.

CONTENTS

CHAPTER 3 : PRESENTING A NEW QUERY SYSTEM

CHAPTER 4 : IMPLEMENTING THE QUERY SYSTEM

INTRODUCTION

Since database management systems have evolved to become a more common management tool for information processing, it is becoming increasingly important that the vast amount of knowledge they process is easily accessible. This fact explains the reason why many people show a deep interest in developping and analyzing query languages.

A query system has been integrated in the ISDOS project, a general software for computer-aided Informations Systems design [PSL.82]. Based on the many years experiments of the Query System by the users and on our personal experience, we have summarized criticisms about the tool and proposals for improvements.

According to these proposals, a new version of the Query System is described in this thesis. Modifications brought to the existing version concern first the functions offered by the Query System (they have been doubled), but also the way the tool is implemented.

The thesis consists of four chapters. Initially, we start with the description of the software environment of the Query System. In addition, a brief overview of query languages in general is introduced. The first chapter describes these two aspects and concludes by a description of the existing QS version.

Before proposing a new version, the existing one is evaluated. We point out the causes of non satisfaction that lead us to propose a new version. Evaluation criteria are defined and used to evaluate the existing version in chapter 2.

Based on this evaluation, a new implementation is proposed and described in chapter 3. This new solution is also evaluated on the basis of criteria defined in chapter 2.

In chapter 4, we present an overview of some aspects of the proposed QS implementation. Its components are described and its realization is discussed in this chapter.

CHAPTER 1 :

BACKGROUND FOR DEVELOPPING A NEW QUERY SYSTEM

1.1. THE DATABASE

1.1.1. Introduction

Unlike other query systems, the Query System described in this thesis is not integrated in a multi-purpose database management system but belongs to a class of special purpose systems, those that facilitate the software development process. This specific purpose infallibly conditions the database content.

The objective of this chapter is to specify the database class to which the Query System (QS) is applicable. However, before going on, it will be useful to define the context of this work. Therefore, a first step will consist of defining the software purpose and explaining the methodology adopted to reach this purpose. A second step will include a more precise description of the system itself and its architecture. We will observe the position adopted by the QS inside the software and the database purpose this QS serves. Thirdly, we will define the database content by studying the meta-model on which the software is based; we should then deduce the set of questions which can be answered using the Query System.

This section includes an Information Processing System example on which will be based all examples troughout this work.

1.1.2. Life-Cycle Support Systems and Informations Processing Systems.

As introduced in sections 1.1.1, the described Query System is integrated in a computer-based system for software development. These computer-based systems are called "Life-Cycle Support Systems" (LSS). As defined in [YUZO.81], a LSS is "a computer-based system that supports activities of a system department in one or more phases of the software life-cycle ". The major functions of a LSS are described as the ability to :

- accept system desciption in some predifined notation
- maintain a database containing the system description
- produce documentation and other outputs based on the system description
- perform control functions of life-cycle activities.

The suggested methodology associated with the LSS to support software development is decomposed in two phases: the LSS generation and the LSS usage to describe an Information System. This methodology is depicted in Figure 1.1 .

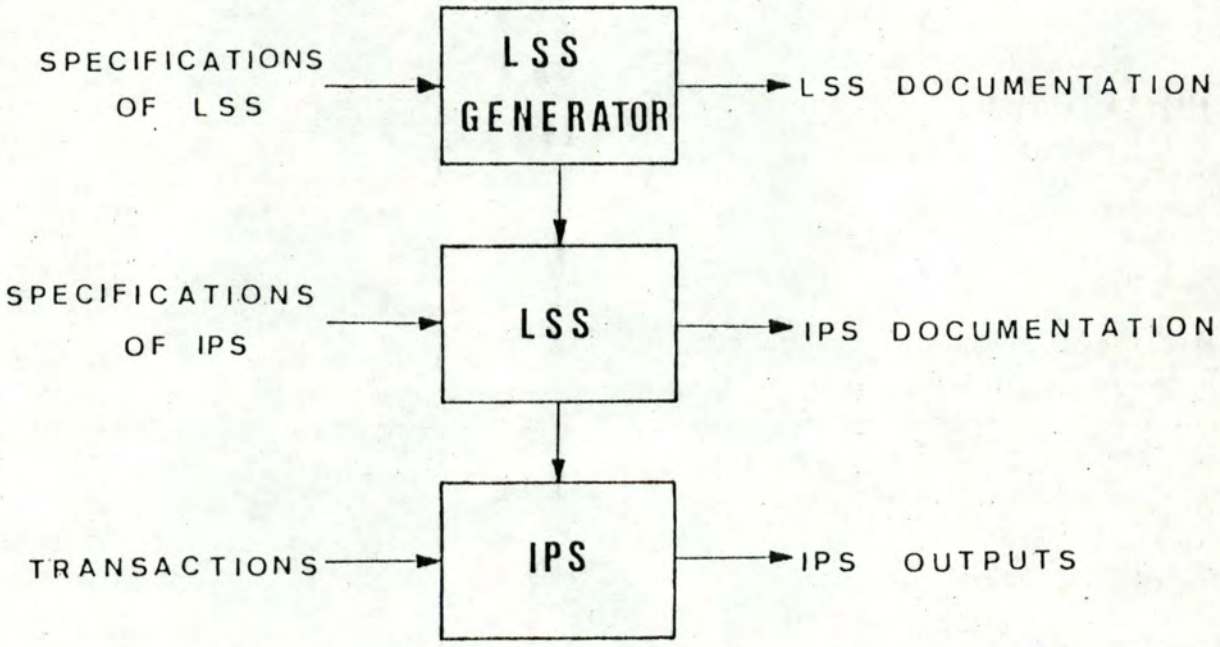


Figure 1.1 from [YUZO.81]

A LSS generator is used to develop a specific LSS. The specific LSS characteristics are determined by the class of the problems we want to describe using this LSS. The LSS generator receives as input the specific LSS description expressed in the LSS description language and generates the corresponding LSS software and its documentation. Basically, the LSS software consists of :

- A language that allows the description of Information Processing Systems whose class has been predefined during the LSS description

- A set of tools allowing to maintain the IPS definition; that includes:

  - a processor for LSS language statements. The processor performs syntactical checks and checks for consistency against informations introduced in the database about the system we want to describe.

  - an analyser to parse the content of the database that contains the specific IPS description.

- a set of tools for updating the information in the database.

- a documentation generator.

In turn, the LSS software is used to generate Information Processing Systems (IPS). An IPS is defined in [TEICH.79] as "the subsystems of the information system in which data is recorded and processed following a formal procedure". These IPS can be manual or computer-based. The term IPS will be used in this thesis to refer to computer-based IPS.

The overall data flow of an LSS is shown in figure 1.2 . The IPS description is introduced in the LSS processor in a predefined format, that is, the LSS language defined during the previous step. The LSS processor analyzes those statements, performs checks and updates the LSS database. This database is a repository of informations describing the IPS; documentation and other reports (such as stimulation) are produced from informations contained in the database.

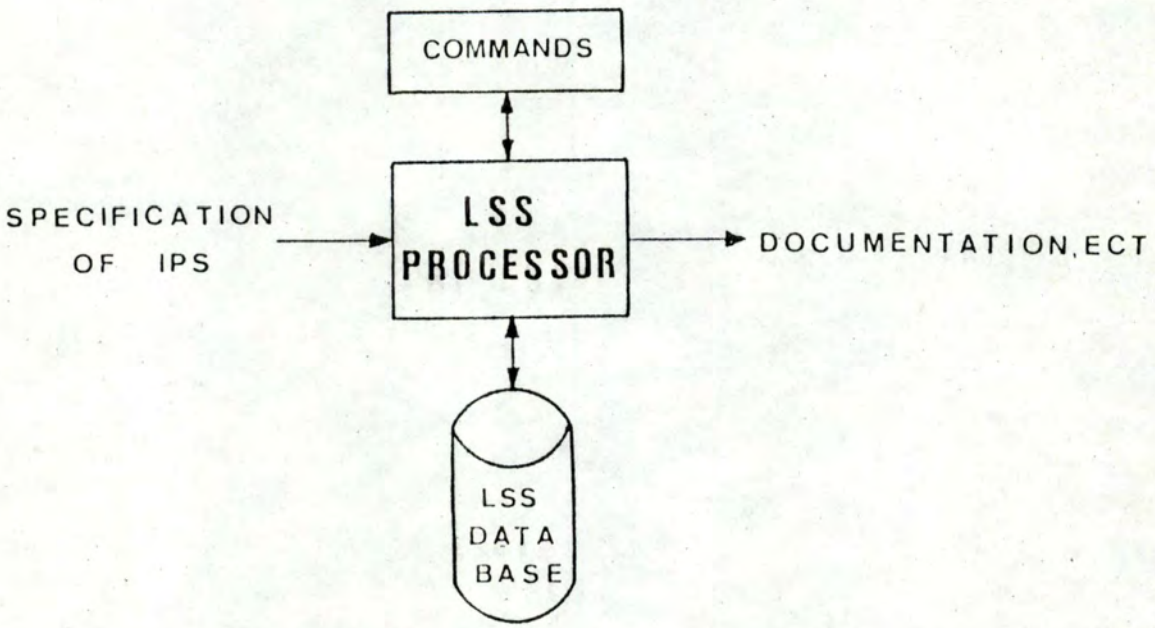


Figure 1.2

1.1.3. ISLDM and SEM : an implementation of an LSS

The Information System Language Definition Manager / System Encyclopedia Manager (ISLDM/SEM) is an LSS generator example. The Query system described in this thesis is integrated in this software. The purpose of the Query System is to query an occurence of the LSS. From this specific purpose, we deduce two underlying consequences: Firstly, the database content queried by the Query System always corresponds to an IPS description. Secondly, the Query System will be determined by this content of the meta-database, namely, the database containing the definition of the IPS description language. At this point, an overview of the SEM structure and the related position of the Query System in the overall software will be instructive.

The ISLDM/SEM structure is shown in figure 1.3 . Following the methodology explained in the preceeding section, the ISLDM/SEM is divided into two parts : the LSS definition and the LSS usage to define a specific IPS.

The first part consists of the specification language definition. A language designer defines a specific Information System Description Language (ISDL) or target language , based on a model defined in the following section (1.1.4). He specifies this language using ISDLM language. The statements expressed in this language are analyzed by ISLDM. It checks that the target language definition is unambiguous, syntactically correct and consistent and produces appropriate error messages, warning ,etc. While it analyzes the specific target language definition, the ISLDM updates a database containing the target language definition. When the language designer is satisfied with his new language, the ISLDM produces tables containing the language description. These tables will be referenced during this new language usage. The ISLDM also produces some reports, including the target language user's manual.

The second part consists of the use of the new target language. The information system under development is described using the target language. The set of statements describing the system are introduced as text with a standard input device or in the shape of a diagram on a graphics terminal. The SEM receives those statements, analyzes them referencing the target language definition contained in the tables generated by the ISLDM. The SEM also updates a database containing the Information System description. This database is called the user-database. Every introduction of new data in the user-database will be preceded by checks that will take into account the target language definition but also the current situation of the database.

We can outline that the SEM is totally independent from a specific target language . The set of functions that the SEM provides to the user does not change from one description language to another one. The reference to a specific language is achieved via tables generated during the target language definition phase.

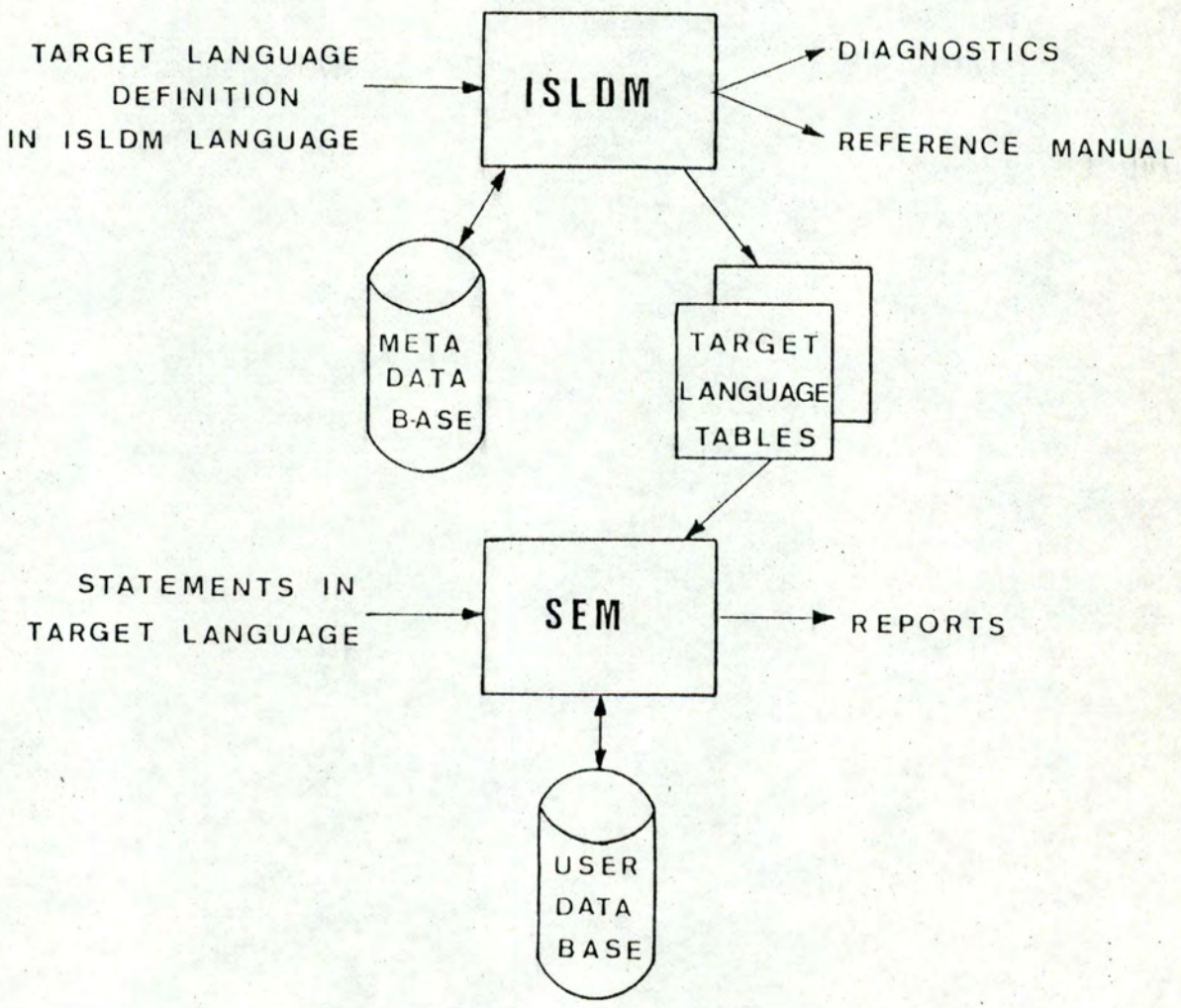


Figure 1.3

Among the functions provided by the SEM, and in addition to the input processor previously described, we can mention the abilities to :

- initialize the database containing the IPS description, before introducing the Information System description.

- update this database (IP : input processor, DP : delete processor, RP : replace processor)

- produce reports (FS: formated statement, STR : structure, EP : extended picture)

- select objects in the database using NS (name selection) or the Query System.

For more informations about these tools, see [YUZO.81] .

1.1.4. Meta Model supporting the SEM - User Database content.

Analyzing the meta-model on which the SEM software is based will allow us to describe the content of the database containing a specific Information System description (or user-database), namely, the main purpose and support of the Query System. Specifying its content, we shall determine what to ask from the user-DB, namely, the set of basic criteria of the QS. This set of selection criteria will be reduced by the purpose assigned to the QS : as a matter of fact, the QS is only designed to select names and output names list that can be used by other tools to produce reports. The reporting fonction is not realized by the Query System.

The meta-model supporting the SEM is essentially based on the Entity/Relationship model [CHEN.76], although different terminology is used. The three major concepts introduced in the E/R model are included in this model with a fourth one added (the constant). Let us describe each concept in detail.

OBJECT

An object is equivalent to the ENTITY notion in [CHEN.76]. An object is a " thing which can be distinctly identified " [CHEN.76]. Its perception as object depends on the administrator point of view [BODA.83]. As such, it is the smallest unit that may be individually created and destroyed [YUZO.81]. For instance, the LSS administrator can define the following objects : a message, a book, a plane,... Each object has certain properties associated with it. Some of these properties are common to all objects in all the LSS; others are introduced by the LSS administrator and qualify specific objects.

RELATIONSHIP.

The relationship is an association between two or more objects or constants , different or not (up to 4). These components involved in the relationship (objects or constants) are called parts of the relationship and the number of parts, the degree of the relationship. The existence of the relatonship depends on the existence of its components [BODA.83] . Each instance of the relationship has the same degree. It is possible to limit the types of objects or constants that assume the role of a part in a specific relationship. One may also determine the connectivity of a relationship, namely, the number of constants or objects involved.

PROPERTY

A property is equivalent to the ATTRIBUTE notion in the E/R model. A property is a characteristic or a quality of an object or a relationship and takes one or more values or value sets. We distinguish properties common to all objects from those that are introduced by the LSS administrator. The first class is called the intrinsic properties. By definition , these properties are predefined. In the existing model, we identify the following intrinsic properties :

a) For objects :

- Object name : Each object has at least one name. One of them, mandatory, is chosen as the main name for this object : the basic name. Other names can be attributed to an object : the synonyms. Nevertheless, all the names (basic name and synonyms) are identifiers. They are called user-names because they are given by a user. Their values are assigned according to the syntactic rules of the LSS language. [WP279] specifies these rules for the ISLDM language.

- Object type : Each object in the user-datatbase has 0 or 1 type. The set of all possible values for this property is system-administrator defined.

- Other specific properties : Other properties are introduced for all objects. They are not essential for the understanding of the system but are useful to manipulate. These include date-of-last-change (DLC), i.e., the date of last modification for a specific object and the number of modification (each time we modify an object in the user-database, its number of modification is incremented by one).

b) For relationships :

- relationship type

- degree : number of components intervening in the relationship

The second set of properties are those defined by the LSS administrator ; they are dependent only on the type of objects. For each property, the set of legal values is defined. This set is called the domain of the property. System-administrator defined properties for a relationship are not considered in this model.

CONSTANT

In addition to information about objects, the database contains constants , namely, objects that represent themselves and have no property. Some constants have a predefined type : integer, real number, string (sting of characters following rules defined by the LSS language) or text (consists of an arbitrarily long sequence of lines; each line is a set of characters). Other constants may be defined by the LSS administrator : they are called name-constant , namely, a symbolic name given to a constant value. For instance, the property "connectivity" could take its value in the following set of name-constants : { 0-1, 1-N, M-N, 0-N }.

As the user-database contains informations about those concepts, we may deduce the different ways to retrieve informations from the user-database using the Query System. The purpose of the Query system is to select object names from the user-database, either to control the database consistency or for further outputs. Properties and relationships are considered as "attributes" about objects retrieved by the Query System and are used by other tools to output reports.

As an illustration, here follows some basic criteria we may use to select objects from the user-database. These criteria are basic in regards to more complex criteria we may build from those basic criteria by using relational operators. Objects may be selected from :

- an intrinsic property :
  - does the object have a specific basic name ?
  - Does it possess any synonym ?
  - does it possess a specific synonym ? (given its value)
  - is the object typed ? not typed ?
  - does it have a specific object type ?
- a LSS-administrator defined property :
  - does it possess a specific property ?
  - does it possess a specific property with a given value ?

- a relationship in which they could be involved :

  - Is the object involved in a specific relationship ?

  - Is it involved with other objects, or a list of other objects playing their specific roles ?

We can outline the need for tools for generalizing , namely, tools allowing to avoid to specify a part of the query . It is worth-while for :

- object names (basic names or synonyms)

  example : names beginning with "A"

- names involved in a relationship, since the degree is constant

- the values for the properties

  examples : 12 THRU 24, <= 12,...

It would be also convenient to combine basic criteria or to specify as part of a query a set of objects selected from another criterion.

example :

which objects are involved in a specific relationship with other objects that have at least one synonym ?

Other tools often proposed by existing query systems such as the average, the maximum, the minimum of properties values are not useful in the context of the Query System described in this thesis.

1.1.5. Example

The purpose of the following example is twofold : to allow a better understanding of this chapter and to support all the examples given throughout this work. More complete examples can be found in [PSL.82] and [DSL.82].

The first step in using this software will consist of the specification language definition. Annex-1 contains the Target Language definition using the ISLDM language. Figure 1.4 presents the concepts intervening in this language. Objects are depicted by a rectangular box. Relationships are represented by a diamond

shaped box with their parts connected by arcs. The connectivity is given on the arc. Specific properties of each concept (objects and relationships) are quoted in the box describing the object. The precise definition of each object is contained in Annex-1.

From the Target Language definition, we may deduct its syntax :

Properties common to each object-type :

```
SYNONYMS ARE synonym-name (,synonym-name) ;

DESCRIPTION ;
     comment entry ;
```

PROCESS section

```
DEFINE PROCESS process-name ;

    PRIORITY { HIGH
             { LOW      } ;
             { MIDDLE
             { NONE

    PERFORMING-TIME  integer ;

    ON TERMINATION TRIGGERS process-name ;

    TRIGGERED BY TERMINATION OF process-name ;

    GENERATES integer TIMES message-name [IF condition-name] ;
```

CONDITION section

```
DEFINE CONDITION condition-name ;

     PROBABILITY real-number  ;
```

MESSAGE section

```
DEFINE MESSAGE message-name ;

    GENERATED integer TIMES BY process-name IF condition-name

    RECEIVED BY { process-name   } ;
                { interface-name }

    GENERATED BY interface-name ;
```

INTERFACE section

```
DEFINE INTERFACE interface-name ;

    RECEIVES message-name ;

    GENERATES message-name ;
```

Once the target language is defined, it may be used to describe a specific information system. Using one of these concepts for description could give :

```
DEFINE PROCESS Book-requisition ;

    SYNONYMS req-ord ;

    DESCRIPTION ;
      a person requests for a specific book by sending
      a loan requisition to the library specifying the
      book title and its number ;

    GENERATES 1 requisition-order IF title-in-file ;

    PRIORITY LOW ;

    PERFORMING-TIME 15 ;

    ON TERMINATION TRIGGERS book-retrieval ;
```

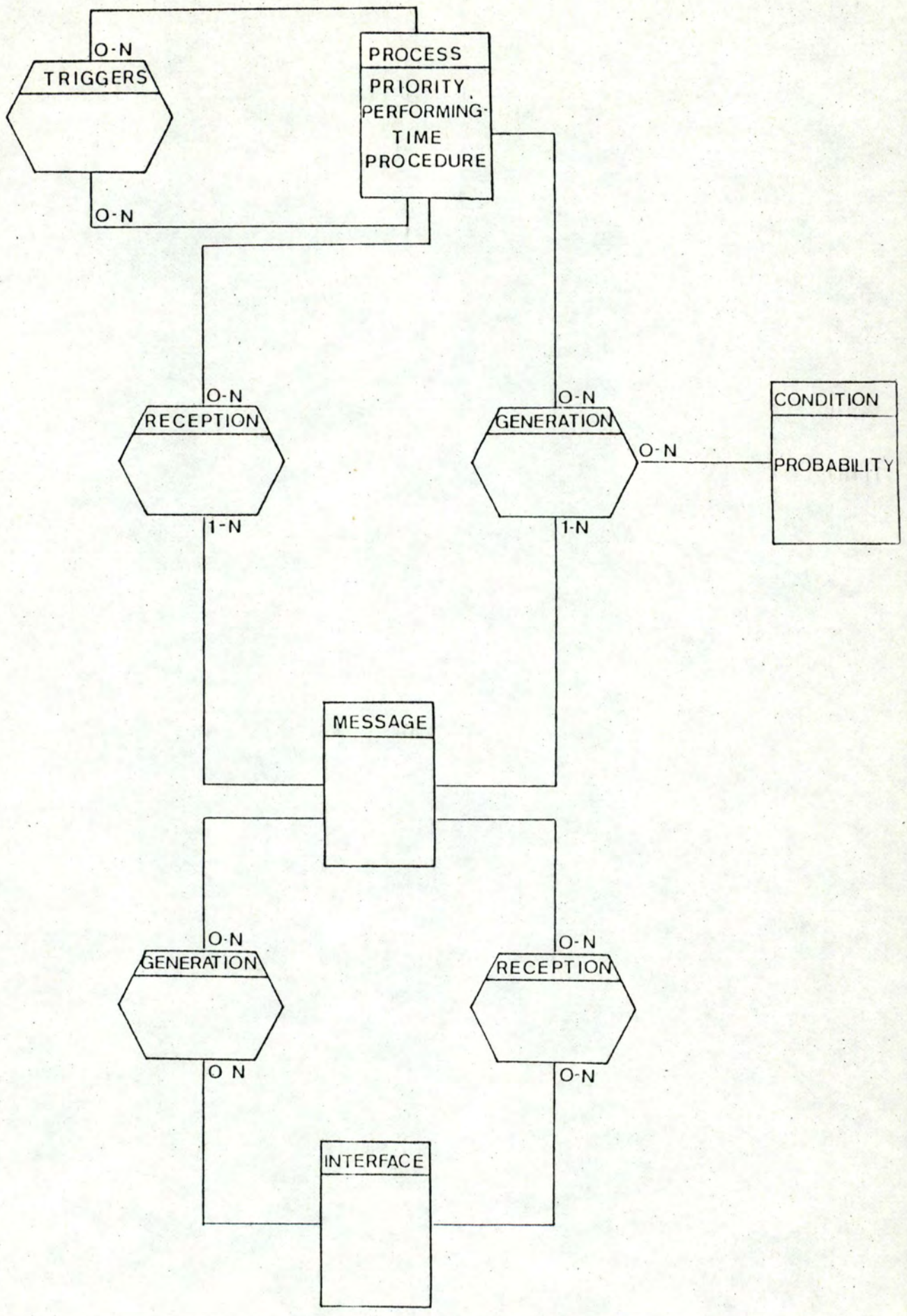

O-N
TRIGGERS
O-N
PROCESS
PRIORITY
PERFORMING-TIME
PROCEDURE
O-N
RECEPTION
1-N
O-N
GENERATION
1-N
O-N
CONDITION
PROBABILITY
MESSAGE
O-N
GENERATION
O N
O-N
RECEPTION
O-N
INTERFACE

Figure 1.4

1.2. QUERY LANGUAGES DEFINITION

1.2.1. Queries and Query Languages

By opting for the database to organize his data, the user decides to associate with his informations a structure that could be relatively intricate. One of the main objectives of organizing a large quantity of data by choosing a database management system is to be able to retrieve any subset of data by specifying the conditions these data must satisfy in order to be selected from the database. The user specifies these conditions using queries. A query in the context of information retrieval is a request for some informations from the database. These questions are stated in a predefined language called the query language.

The program which allows the retrieval of informations from the database according to queries is named the Query System. The QS analyzes a query expressed in the query language, selects from the database data satisfying the conditions expressed in the queries and output this data in a way that can be easily understood by the user. Unlike the Query System described in this thesis, most Query System encountered in the litterature allow the user to retrieve data but also to modify, to insert and to delete them.

The query languages are generally dedicated to be used independently of a procedural programing language. The lack of technical knowledges on the part of potential users incites QS designers to the user-friendliness of those system : the request expression is facilitated by using syntaxes looking as much as possible to "natural" expressions. (SEQUEL [CHAM.76]) ; by allowing th user to express his queries by giving examples (QBE [ZLOO. 80]); by allowing a step by step experiment : what the user knows about the language is just what he needs; by adding graphic interfaces (table representation in QBE or graphic representation of objects in Spatial Management of Data [HERO.80]). As summary, two golden rules must be followed in designing Query System : ease-of-use and ease-of-learn.

Not all question can be expressed within a query language : we distinguish open questions from closed ones. Open questions, such as "what do you know about computers ?" have no clear-cut answer. With respect to a database, a complete answer may be a durp of all data. Open questions are therefore excluded from the query languages. On the contrary, closed questions have clear and, given a database, unique answers. For example, "Does XYZ make computers ?" or "What are the models made by XYZ ?" are closed questions. The first one is a "yes-no" question, the second one, a "list" question. Those questions can be interprated according to a specific database.

1.2.2. Query Language classification.

A first classification could be based on the data model on which those languages are based : a query language is designed in a different way if it is based on the relational model, hierarchical model or network model. Even if this fact plays an important role in the query language design, it does not settle a classification element by itself.

Another classification could be based on the way the information is manipulated : in fact, two different approaches allow the user to express his queries. The first one, named set languages , allow him to state in some algebraic form the rules which relate a new set of data to existing ones. The second one requires a user to state the conditions which need to be met for an object from an existing set to belong to a new one : they are the languages based on predicates. Let us determine the characteristics of these two query languages classes and settle which class the Query Language described in this thesis belongs to.

1.2.2.1. Predicate languages

The predicate languages allow the user to express the condition an object must satisfy in order to be selected, based on the predicate calculus. By referencing this well-known symbolism used in other contexts, it allows to avoid a new semantical definition given to these expressions.

A predicate , in the context of information retrieval could be seen as an expression taking the value "true" or "false" for each element from the database. A predicate identifies a set of individuals who have a certain property. The property is expressed by a relation between attributes and values or between attributes.

Example :

> The predicate (AGE = 18) identifies people whose attribute 'age' possess the value '18' .

A predicate can be expressed in a mathematical-like form or close to "natural" expressions. For memory, following elements can be used in a predicate :

- boolean operators

  example : (SEX = 'MALE') AND (AGE >= 21)

- quantifiers : SOME, ALL

  examples :

  - select departements in which some woman is working
  - select students whose grades are "A" in all courses

- functions that calculate new values from existing attribute values (sum, minimum, maximum, average, ...)

  example :

  - select employees who earn more than their departement average.

- implication, even if it is redundant

Three modifications are necessary to transform the applied calculus into a query language : give an interpretation to its free variables, namely, the variables not preceded by a quantifier; define the particular predicates available and link the universe of the calculus to a specific database.

Most of the time, a predicate is embedded in a more complete expression to form a query. A question in the predicate language can be decomposed as :

‹target-list› ‹predicate› ‹order-expression-list›

where :

‹target-list› : defines the information types we want to be selected from the database.

example : name and age of a person

‹order-expression-list› : allows to specify the order in which selected data must appear

example : select customers by city;
products by increasing number.

The predicate languages are the most common ones. Among the most well-known ones, we can mention SEQUEL [CHAM.76], QBE [ZLOO.80], QUEL [STON.78] and FQL [PIRO.74] .

1.2.2.2. Set languages

The set languages allow the user to state in some algebraic form the rules which relate a new temporary set of data to the existing sets. A language belonging to this class consists of a collection of operators. They are :

Union :

The union of two sets A and B is the set of all objects belonging to A or B or both.

example :

A = customers who have already ordered today

B = customers who have still something to pay

$A \cup B$ = customers who have ordered today or who have still something to pay.

Intersection

The intersection of two sets A and B is the set of objects belonging to A and B.

example :

A = persons who are customers

B = persons who are suppliers

$A \cap B$ = persons who are customers and suppliers

Difference

The difference between two sets A and B is the set of objects belonging to A but not to B.

example :

A = persons who are customers

B = persons who are suppliers

A \ B = persons who are customers but not suppliers.

Complement

The complement of set A is the set of objects that do not belong to A.

example :

A = persons whose sex is "male"

~A = persons whose sex is "female"

Cartesian product

A * B = set of objects which are built by a concatenation of an object belonging to A and one belonging to B.

example :

A = product numbers

B = customer numbers

A * B = set of tuples built by taking a product number and a customer number.

The number of operators is settled by the usage assigned to the specific query language. Moreover, operators specific to the data model can be introduced (for instance, projection or join operator for the relational model).

The set languages class is not the most common one. TABLET [STEM.78] constitutes an example. The Query Language described in this thesis belongs to this family.

1.2.2.3. Procedurality and other criteria

To compare these two languages classes in terms of simplicity or extension possibility is not easy and meaningful because these criteria can only be evaluated against a specific language. However, we can compare them from a procedual aspect.

A non-procedural query, according to Date [DATE.77], p 84, "states merely what the result of the query is, not how to obtain it". There is no doubt that a set language is more procedural because it "specifies a step-by-step method for achieving a result" [WELT.81]. The query expression specifies in which order the operators must be applied to the existing sets and in which order those sets must be created. On the other hand, a language based on the predicate calculus is less procedural.

People assert that the procedurality is prejudicial in order to facilitate the use of the language because the user looses time to choose the best way to state his query. At the other hand, studies comparing a more procedural language, TABLET with a more non-procedural one, SQL, [WELT.81] and [REIS.81] try to prove that "people more often write difficult queries using a procedural query language than they do using a non procedural query language" [WELT.81]. As far as we are concerned, the language ease-of-use must be based more on its syntax,its interface with the user, its step-by-step learning or its execution speed than its procedural aspect.

1.3. FUNCTIONAL DESCRIPTION OF THE EXISTING QS VERSION

1.3.1. Introduction

We have established in chapter 1.1.3 the position of the Query System in the general software. We have defined its purpose : to supply a list of user-names that satisfy a specified criterion. Such a list may among other possibilities be produced in a format that can be directly used as input to other SEM commands. Besides the selection function, the current QS version supplies a set of tools that facilitate the selection. The purpose of this section is to present an overview of the functions of the existing Query System implementation (selection functions and other functions). Initially, we define the two main concepts used in this QS : the set and the criterion. Next, we list the possible basic criteria which can be used in the building of a more complete criterion. Finally, we give a summary of the commands available in the existing QS version.

1.3.2. Basic concept definitions

1.3.2.1. the set

A. Definition

A set is defined as a collection of objects selected from the user-database. The selected objects must satisfy the criterion defining the set. A name can be assigned to a set if the usage justifies this name. This name identifies the set from other ones.

B. Usage

- To list a set, namely, to output the list of objects belonging to the set; the list can be obtained by invoquing the set-name or by giving its definition (a set-name is not mandatory in this usage).
- To be referenced in building other sets : in this case, it is necessary to identify the set by using a name.

C. Specification

The set definition is introduced by using a criterion.

1.3.2.2. Criterion

A. Definition

The criterion defines a set : it express the conditions an object must satisfy in order to be selected from the user-database. A name can be given to a criterion, depending on its usage.

B. Usage

- to define a set; no criterion-name is mandatory.
- to be used in the definition of a set or another criterion; in this case, a criterion-name is mandatory.

C. Specification

a criterion is specified as a logical expression where :

- the operators are :
  - AND : intersection
  - OR :union
  - NOT : complement
- the operators are :
  - a basic criterion as defined in section 1.3.3
  - a set-name
  - an other criterion name
- parenthesis can be used to modify or to emphasize the operators priority rules.

examples :

if C2 and C3 are names of criteria previously defined, a new criterion named C1 can be defined as :

C1 = C2 OR C3 ;

C1 = C2 AND (BN = A ?) ;

C1 =(BN = A ?) AND NOT HAS SYNONYMS ;

since [BN = A ?] and [HAS SYNONYMS] are basic criteria defined in section 1.3.3 .

1.3.3. Basic criterion definition

The list of basic criteria is presented in this section. A natural English formulation of the inquiry will be given followed by its translation in Query Language. The basic criteria are gathered by the object of the inquiry. This list corresponds to the list of possible queries given in section 1.1.4 .

A. possession of name

A.1 Does the basic name have a specified form ?

BN = ‹ match-string ›

where ‹match-string› is any user-name in whole or in part with question-mark(s) used to mark unspecified parts of the name

examples :

BN = book-retrieval-in-library ;

BN = book-retrieval- ? ;

BN = ? - retrieval- ?

A.2 Does the object possess synonyms ?

HAS SYNONYMS

A.3 Does the object possess at least one synonym of a specified form ?

SN = ‹match-string›

B. Possession of a date of last change

B.1 Does the object date of last change lie within a specified range ?

DLC ‹range-specifier› ‹date›

where :

‹range-specifier› is in (=, ‹, ›, ‹=, ›=)

‹date› has the following format :

(MM-DD-YY or MM-DD-YYYY)

example :

DLC ‹ 02-27-1982

any object that has been changed before february the 27th 1982.

B.2 Does the object's date of last change occur before (after) a specified change sequence number ?

```
MAXC = ‹integer› ;

MINC = ‹integer› ;
```

example :

MAXC = 3 ;

any object which change sequence number is lower or equal than 3.

C.Possession of a type

C.1 Does the object have a type ?

```
TYPED
```

C.2 Is the object without a type ?

```
UNTYPED
```

C.3 Does the object have a specified type ?

```
‹type-name› : this name is defined in the
              target-language.
```

example :

PROCESS ;

any object which type is "PROCESS".

D. Possession of the various properties defined in the target-language

D.1 Does the object have a specified property ?

HAS ‹property-name› : this name is defined in the target-language

example :

HAS CONNECTIVITY ;

any object that possess a connectivity.

D.2 Does the object have a specified property with a specified property-value ?

HAS ‹property-name› WITH ‹property-value›

example :

HAS PROBABILITY WITH 0.25 ;

any object that has a probability of 0.25 .

E. Possession of textual comments

E.1 Does the object have a textual comment of a specified type ?

HAS ‹textual-commemt-name› : this name is defined in the target-language

example :

HAS DESCIPTION ;

any object that possess a description.

E.2 Does the object have any textual comments ?

HAS TEXT

F. Involvement in relation occurences

F.1 Is the object involved in no relation ?

ISOLATED

F.2 Is the object involved in a specified relationship with specified objects ?

‹object-designation› ‹statement-form›

where : ‹statement-form› is constructed by taking any legal statement from the target-language and substituting an ‹object-designation› wherever an object-name may appear.

‹object-designation› must be one of the following :

- a user-name
- any of the above basic criteria (from A. to E.)
- a pre-defined set-name
- one of the above preceded by the symbol "~"or by "NOT" : those symbols are used to signify the complement

  ex : NOT PROCESS = any object which type is not PROCESS

- the symbol "!" to refer to any object in the user-database
- the symbol "?" that refers to those objects in the user-database we are looking for on and only one question-mark must be used in any criterion of type (F.2) .

examples :

- ? RECEIVED BY Book-reader ;

  any object received by a "Book-reader".

- NOT (INTERFACE) RECEIVES ? ;

  any object received by something else than an interface.

- ? GENERATED ! TIMES BY ! IF ! ;

  any object that possess the quoted sentence

1.3.4. User interaction

The existing QS implementation explicitly brings to light the distinction between the criterion and the set , concepts defined in section 1.3.2. Moreover, commands specific to these two notions are available (CRITERION command for the criterion and SET command for the set). These criteria and sets of user-names satisfying a criterion can be named and stored for later use.

By distinguishing the two concepts, we can formulate the question in two different ways :

- list the names of objects which satisfy a specific criterion (LIST command).
- check if a list of objects satisfies a given criterion (CHECK command).

The output furnished by the QS may be expressed in two forms :

- A report,in a specific format with the set-name,the text of the criterion defining the set and the list of objects satisfying the criterion.
- A report,in the shape of a file that can be used again by other SEM commands. This file contains a list of user-names, one name on each line.

Two modes of operation are possible for using the QS : interactive way or batch mode with user absent. In the first way, the commands are typed in one by one from the user's input device, and the user will be consulted for error correction. In the batch mode, the commands are read from the input streams and if any error is detected, the Query System itself will decide whether to attempt a plausible correction, continue assuming that the error has only local effects or abort.

example :

a false name-specification for LIST command will have as effect to request for a meaningful specification in interactive mode; if the command is entered in batch mode, the command is ignored.

```
> LIST S1 ;
  name S1 is not defined before.
  enter replacement.

> SET1 ;
    .
    .
    .
```

1.3.5. Command Summary

Functions of the Query System are available via commands. These commands allow the user to define sets and criteria, output results and manipulate those objects. We procede to give the name and the purpose of each command. For further informations about the syntax of the commands or about examples, see [TM119] and [WP456].

A. Set and criterion definition

command-name	purpose
SET	to select a set of objects from the user-database and name the selected set.
CRITERION	to name a criterion and to refer to the criterion by the name in the later commands.
READ	to read a list of object-names or a criterion from a file.

B. Result outputs

CHECK	to check if a set of objects satisfy the criterion and list informations. Three parts of information are generated by the command : summary information, satisfied list and unsatisfied list.
LIST	to display a set of objects which satisfy a criterion.
PUNCH	to punch out a list of object-names or a criterion to a file.

C. Set and criterion manipulation

CHANGE	to change user-defined set-name or criterion-name
DISPLAY	to display a previously defined criterion or to make a list of set-names or criterion-names defined so far
EXPLAIN	to make an explanatory comment on the command specified

D. Others

STOP	to terminate the session with the Query System.

CHAPTER 2 :

CRITICAL EVALUATION OF THE EXISTING QS VERSION

2.1. PROBLEM IDENTIFICATION

The observation of ISDOS users and its Query System in particular leads us to the following conclusions : the users fail to utilize the Query System as often as hoped by designers. Besides, they omit to use the tools provided in the Query System to facilitate a further interpretation of the queries. Let us try to clarify the reasons of this misuse by investigating the problem first from the user's point of view and then from the designer's view point.

Let us try to outline the causes for user scepticism towards the Query System under its present version. Four main causes can be brought to light. Their importance vary from one user to another. Therefore, they are presented without any specific classification.

The first cause concerns the performance obtained in executing a query. The user must be quite patient to wait for the answer to a query. While this tool is designed to be used preferably in an interactive mode, the Query System does not allow, for instance, the user to execute consistency tests in an interactive mode because of the time it takes to run these tests.

A second reason is that all the work performed during Query System session is irremediably lost once this session terminates even though the user needs to ask the same queries after modifying the user-database , or simply wants to interrupt the session and to continue it later without having at first to reintroduce what he has done before.

The third reason deals with tools available in the existing version. Some of them are needed and fail (e.g., there is no way to select the objects whose type is "condition" and whose probability is greater than 0.900). Other tools would facilitate the sets manipulation.

Finally, the memory space that can be used during a session is predefined (installation dependent). If the QS spreads in time, we can easily reach this space limit, even though some sets or criteria built during this session are now useful. They could be removed and they would allow to find back the space they used.

Besides, there is an added difficulty in using correctly SET and CRITERION notions and distinguishing the differences between these two concepts. Some syntactic rules seem to be misunderstood and need modifications or a better explanation (user's manual) .

In trying to improve the present version of the Query System, it is appropriate to locate the source of these problems. Most of them are located in the Query System itself and can be solved by modifying this part of the software. However, if we want to decrease the time needed to interprate a query, modifying the Query System does not solve the entire problem : other parts of the software used by the Query System must also be taken into account. Among them, we can mention the user-database manager : its organisation requires to scan sequentialy the entire user-database for most of the queries evaluation. We also think about the error messages subsystem : because of its implementation, the user realizes that he has made a mistake, not thanks to the error message displayed on the screen but because he has to wait for an answer from the system longer than usual.

Before suggesting improvements to the existing version of the Query Sytem, a critical appreciation of the existing implementation will be helpful. This evaluation will be realized on the basis of the criteria previously defined. Section 2.2 contains these criteria. They will also be used to evaluate the new implementation proposed for the Query System.

2.2. EVALUATION CRITERIA

The criteria listed underneath will allow us to evaluate the Query System in its environment as specified in the chapter 1. These criteria relate to the tool and the possibilities it provides but also to the tool's implementation. These two aspects will be successively discussed in the following section.

2.2.1. Functional aspect

2.2.1.1. Language Intrinsic Qualities

[C1] language user-friendliness :

This criterion comprises four aspects :

1. In our context, a database query system is above all composed of keywords. These keywords should express simple concepts, significant for the user.

2. Tools should be integrated in the Query System in order to facilitate the expression of the query. Many implementations can be designed to achieve this purpose : a "HELP" command explaining the purpose of the command and its syntax; a set of menus presenting all the alternatives proposed to the user and a way to select them (numbers, joystick,..).

3. The query language should be consistent with the existing languages from other parts of the general software. In our case, we are refering to the Target Language and to commands languages available in the Input Processor, and in report tools (FS, EP, ..). With regard to these languages, the choice of keywords in the query language must be consistent with those of these languages.

4. The query language should allow an easy formulation of the queries. That includes, in case of intricate requests, a mechanism which allows the user to split up his queries in many other ones easier to interprate.

[C2] Intrinsic definition of the basic concepts and non redundancy :

The definition of the basic concepts (criterion, set and range) should be independent from the context in

which they are used. These basic concepts shoudn't be semanticaly redundant.

2.2.1.2. Functions offered by the Query System

[C3] Commands funtionality :

The purpose of the functions offered in the system is to facilitate the basic concepts manipulation in constructing queries and in presenting the results. These commands should be simple and non redundant. Their numbers and purposes will depend on the user requirements and on the applications to which this tool is intended.

2.2.1.3. System Usage

[C4] QS response time :

the time needed to execute a command or to interprete a query must be "short enough" to save the interactive aspect of the tool.

[C5] Usage in different modes :

The system could be used in interactive mode but also in batch mode. A dialogue with the user could be provided but must be supplied by a decision taken by the program itself in batch mode.

[C6] Queries classification :

In the context of specification database, the queries classification is often helpful. Indeed, the user often wants to gather in a same class a set of queries dealing with a same aspect of the Information Processing system described using the target language. For instance, we could gather queries that deal with the model consistency. This class of queries is defined once for all and called each time we want to check the consistency of a model.

[C7] Functional integration in the software :

In the context of a software for computer aided Information System design, this criterion comprises three aspects :

- Integration of the specific tools : they should manipulate the same data, namely, the description of the Information System.

- Independence of the tools : they should be designed to be used separatly.

- Complementarity of the tools : the results of a tool should be easily communicated to another tool without any interfacing problem.

2.2.2. Implementation aspect

[C8] program maintenability :

The program must be easily modifiable by adding new commands or new basic concepts. Therefore, the two following principles are essential : the program modularization and the independence between the program and its internal data structure : every functional modification can have important repercussions on this structure and indirectly on the entire program if the program and the internal data structure do not communicate via an interface.

[C9] independence towards supported languages :

The tool must be independent from the languages it supports. A syntactical modification should have no consequence on the program itself. This aspect is perfectly achieved for the general software (and then the Query System) with regard to the target language but it must also be verified for the Query System and its Query Language : the part of the program that interprates the requests expressed in the Query Language must be as independent as possible from the other parts of the program.

[C10] Integration in the software during the implementation :

The integration within the general software is also important : a modularization can be achieved between many tools in the software; a part of a specific tool can be integrated in another tool to facilitate the maintenance.

2.3. CRITICAL APPRECIATION OF THE EXISTING VERSION

Let us use each criterion described above to evaluate the existing version of the Query System. References between square brackets refer to the corresponding criterion in part 2.2 .

2.3.1. Functional aspect

[C1] language user-friendliness :

1. Once we consider the Query System purpose and the set of selection classifications (properties and involvment in relationship), the syntax for the basic criteria have been chosen in a significant way. To express them, the keyword notation has been chosen (such as SQL) rather than the positional notation (such as SQUARE). However, we cannot say that the language is user-friendly : indeed, it requires the user to know the target language in detail even if its implementation fail to facilitate its expression. (four corners rules).

   example :

   in the DSL language [DSL.82], to identify any SYNCHRONIZATION-POINT that is not contributed by any event, the query is :

```
SET S1= SYNCHRONIZATION-POINT AND NOT
  ( [? CONTRIBUTED BY GENERATION OF ! ] OR
    [? CONTRIBUTED BY GENERATION OF ! IF NOT ! ] OR
    [? CONTRIBUTED BY TERMINATION OF ! ] OR
    [? CONTRIBUTED BY TERMINATION OF ! IF NOT ! ] OR
    [? CONTRIBUTED BY INCEPTION OF ! ] OR
    [? CONTRIBUTED BY INCEPTION OF ! IF NOT ! ] OR
    [? CONTRIBUTED BY REALIZATION OF ! ] OR
    [? CONTRIBUTED BY REALIZATION OF ! IF NOT ! ] ) ;
```

   even though the user would like to express his query as :

```
SET S1 = SYNCHRONIZATION-POINT AND

NOT [? CONTRIBUTED BY !] ;
```

whatever are the statements occuring in this question.

2. A HELP command is available in the existing QS version. It provides either a list of all the existing commands or detailed explanatory comments about a specific command. These comments allow the user to understand the command purpose and to find back its syntax.

3. The existing Query Language is consistent with the other languages of the general software : The Target Language is integrated in the Query Language within the queries expression.

4. The mechanism of decomposing an intricate query into many other ones easier to interprate is one of the basic principles on which the Query System is based. This mechanism is realized in many query languages by introducing the "variable" notion. In the Query Language, it is introduced with the SET and CRITERION concepts. Its usage may even be stimulated in case of intricate queries.

[C2] Intrinsic definition of the basic concepts and non redundancy :

No meaningful difference exists between the SET and CRITERION notions. Moreover, the explicit distinction between these two notions introduces some confusion for the user. This distinction was only justified because it allowed us to use many times the same part of a criterion in the definitions of sets or other criteria. Since a set-name may be used in constructing other criteria, the CRITERION command and the underlying concept may be removed from the Query Language without any loss of functional capacity of the Query System.

[C3] commands functionality :

There is no way to settle the ideal set of commands because its number varies from one user to another. On the other hand, a high number of commands would be prejudicial to the understanding and the mastering of the tool. However, we propose to introduce the following commands that allow :

- to save and to restore sets from a session until another one starts in a special purpose database.

- to specify a set of values for a property or a constant intervening in a statement.

- to control the extent to which the queries evaluation is done. According to the wishes of the user, the query can be evaluated from the entire database or a subset of this database.

- to document the sets definitions and to facilitate their interpretation by other users.

- to delete unuseful sets and to retrieve the space they employ.

- to execute commands contained in a file.

- to use control structures in the command definition. As proposed by Kang [TM427], the following structures may be introduced :

```
1)  FOR EACH ‹variable› of ‹set-name› DO
                          .
                          .
                      queries;
    END;

2)  DO UNTIL ‹condition›;
                          .
                          .
                      queries;
                          .
                          .
    END;

3)  IF ‹condition› DO ;
                          .
                          .
                      queries;
                          .
                          .
    END;
```

where the following operands may be used as ‹condition› :

1) CARD ‹set-name› (› | ‹ | ›= | ‹= | ‹› | =)

(‹integer› | CARD ‹set-name›) ;

: returns cardinality of a set.

2) ‹attribute-name› (› | ‹ | ‹= | ›= | ‹› | =)

‹value›

this last possibility is only justified in the case of an integration of the Query Language in a host language.

[C4] QS response time :

The execution time of a query is in our opinion, one of the weakest aspect of the existing QS version. This time rapidly increases with the increase in the user-database dimension because most of the queries evaluations require in the existing version a complete scanning of the user-database.

[C5] Usage in different modes :

In the existing version, commands may be introduced either in interactive mode or in batch mode. If the second mode is chosen, in case of reaction by the user in the interactive mode, the program itself will decide if it is convenient to attempt a plausible correction, to continue with the assumption that the error has only local effects or abort the command.

Example :

in interactive mode :

```
› LIST books ;
  name 'books' is not defined before ;
  enter replacememt.

› Books ;
  too many errors.
```

in batch mode :

```
› LIST books ;
  command ignored.
```

[C6] Queries classification :

In the existing version, there is no way to classify queries. The three following implementations can be realized (*).

1. Under the shape of an external file containing queries and other commands; by calling the file name, we will execute all the commands contained in this file. However, this implementation is more a way to facilitate the queries introduction than a real classification.

2. A new concept named CLASS is introduced. A class is defined as a set of queries built by the user. Instead of manipulating individually the queries, the user will refer to the class.A class is defined using the following command :

CLASS ‹class-name› [CONTAINS] ‹list-of-set-names› ;

The class can be saved in a special purpose database ; it can be restored from it and listed.

example :

```
SET S1 = UNTYPED ;

SET S2 = ISOLATED ;

SET S3 = MESSAGE AND NOT ( ? RECEIVED BY ! );
```

A class named "CONSISTENCY" will be built; it will contain these three queries that check the model consistency :

```
CLASS CONSISTENCY CONTAINS  S1,S2,S3;
```

3. A pseudo-macro can be defined and possibly parametrized.

example :

We could define a set of macros to evaluate the model consistency. These macros are defined once for all, "pre-compiled" and saved. They can be called by specifying the object-type under study (parameter "obj-type").

(*) A command to classify queries already exists in the DSL/DSA version in the University of Namur

```
MACRO CONSISTENCY (obj-type);

    SET S1 = 'obj-type' AND ISOLATED;

    LIST S1;

    SET S2 = 'obj-type' AND UNTYPED;

    LIST S2;

    IF ('obj-type' = MESSAGE)

        SET S3 = NOT (? RECEIVED BY !);

        LIST S3;

    FI;

    .
    .
    .

END;
```

This last implementation has two additional advantages : a macro can be parameterized and can include control structures. However, at our opinion, this last solution is too comprehensive and intricate in this specific context.

[C7] Functional integration in the software :

Let us inspect the three aspects of this criterion :

1. The tools are integrated : all the tools are working on the same database containing the described Information System : the user-database.

2. The tools can be used separatly : They have specific purposes and can be used autonomously to achieve this specific purpose.

3. As the report function and the database update are not integrated in the Query System, the result of a name selection using the QS needs to be transmitted to other tools. The transmission is achieved via a file containing the list of user-names. No modification need to be done to communicate this list from the QS to another tool.

2.3.2. Implementation aspect

[C8] program maintenability

The program must be designed in such a way that any functional extension may be realized without major modifications of the existing program. The moduralized design is one of the basic implementation concepts of the ISDOS software and of the Query System in particular.However, in the existing version, the independence between the program itself and its internal structure is not sufficiently realized : every modification of this structure will have repercussions in the entire program. For instance, the following modification will require other modifications in all the subroutines that access the internal data structure :

> the definition of all the sets are contained in a unique list in the existing version; we want to divide this list in smaller ones (one for each set) to use efficiently the memory space.

This aspect is even more important if we forsee new extensions while a first implementation is proposed.

[C9] independence towards supported languages :

From a logical point of view, there is no connection between the query language and the program that answers to the queries expressed in the language. Those two parts can be designed autonomously and can communicate via an interface. Thereby, no syntactical modification will have repercussions on the program itself. In the existing version, the syntactical analyser could be easily isolated from the rest of the program but every syntactical modification requires to update tables and block data containing the language definition. We need to find a better way to create and modify this language.

[C10] Integration in the software during the implementation

The syntactical and semantical analysis for a statement is already provided in the Input Processor (IP), another part of the general software. This analyzer is called "SNTX". Eventhough the IPS's purpose differs from the purpose of the Query System (the IP is designed to introduce objects definitions in the user-database), SNTX can be modified and integrated in the Query System.

CHAPTER 3 :

PRESENTING A NEW QUERY SYSTEM

3.1. INTRODUCTION

Given the remarks formulated in the last chapter as regards to the unsuitability of the existing version of the Query System, a new solution is proposed and described in this chapter. In so doing, we shall try to meet as much as possible the objections formulated in the preceeding chapter. It should be noticed that the proposed version constitutes only a partial one among others.

A new version is required because of the need for functional improvements, namely, a lack of tools and a bad comprehension by the user of the basic concepts. The solution described in this chapter will answer mainly to these criticisms. Its improvements will be realized by additional commands but also by proposing a methodology in constructing a query. We also took advantage of the opportunity to improve the implementation aspect of the Query System.

This chapter will begin with the list of improvements and modifications introduced in the new version. This version will also be evaluated in this chapter, basing on the criteria defined in section 2.2 .

3.2. IMPROVEMENTS AND MODIFICATIONS

3.2.1. Functional level

3.2.1.1. Criterion Concept Removal

In the existing version of the Query System, the SET and CRITERION concepts are implemented by specific commands even though there is no semantic difference between the two notions : the criterion is nothing else but the expression that allows to specify the properties that objects must satisfy in order to be selected from the user-database.

Associating a specific command to the criterion has only one advantage : to avoid to duplicate or to reintroduce a part of the set definition. As it is possible to define a set from other sets previously defined, the possibility to create a hierarchy of sets still remains; using this hierarchy, we can avoid a multiple introduction of the same parts of the criteria.

In the new QS version proposed in this chapter, the command that allowed to define a criterion is removed (CRITERION command). Henceforth,the criterion concept will never exist anymore independently of the set notion : it won't be possible to name a criterion. As summary, the following concepts will be associated to a set in the new version :

- A criterion, namely, the set definition; it express the condition that the objects must satisfy to be selected from the user-database.
- A list of object-names satisfying the criterion.
- Possibly comments lines that help the criterion comprehension or that specify the universe in which this criterion was evaluated (see section 3.2.1.5) .

Because of the removal of the criterion as an independent concept, some modifications are introduced in commands syntax (without modifying their purposes). A comprehensive description of these commands can be found in [IM47] .

3.2.1.2. Range Concept Introduction

3.2.1.2.1. Presentation

In the existing version of the Query System, there is no way to specify a set of values for a property or a constant that occurs in a statement. This constraint is not realy restrictive when the values taken by these properties or constants are integers or name-constants : we can enumerate the values, even if this method is tedious. Nevertheless, if those values are real numbers or strings, the need for a new command is still more justified because there is no way to express queries with these types of values. The two following examples illustrate requests in which a sequence of integers and of real numbers occurs respectively :

examples :

- Select process that generate between 1 and 10 messages.
- select conditions whose probability is greater than 0.850 .

With the introduction of the range command and the underlying concept, this problem is solved.

3.2.1.2.2. Range Concept Defintion

A. Definition

A range is a set of constant values; a name is assigned to this set, identifying this range among other ranges.

B. Usage

To introduce a constraint on constants contained in the user-database. Wherever a constant may occur in a query, it can be subtituted with a range. Mainly, this could happen in the two following cases :

- a value for a property

examples :

PROBABILITY 0.15

PRIORITY LOW

- Constants occuring in a statement :

example :

GENERATES 5 TIMES msg-1 IF cond-1 ;

The queries expressed above can be solved in the following way :

- "Select conditions whose probability is greater than 0.950"

a) a range is set whose values are greater than 0.950 (using RANGE command) :

RANGE R1 = GT 0.950 ;

b) the query is expressed by substituting the range-name 'R1' with the constant :

SET S1 = HAS PROBABILITY R1 ;

- "Select process that generate between 1 and 10 messages" :

a) a range is set whose values are included between 1 and 10 :

RANGE R2 = 1 THRU 10

b) the query is expressed by substituting the range-name 'R2' with the constant :

SET S2 = [? GENERATES R2 TIMES BY ! IF !] ;

Commands available for the manipulation of sets are also available for the range (DISPLAY,ERASE,SAVE,...) .

C. Specification :

Using RANGE command.

3.2.1.2.3. Range command

A. Purpose

To define a new range, namely, a set of values that can be finite or infinite.

B. Syntax

```
RANGE <range-name> = ( <range-specification> |
                       "{" <set-of-values> "}" ) ;
```

where :

<range-name> is a string of maximum 30 characters following ISDOS conventions. The name is identifier between the range-names and the set-names defined so far during the current QS session.

<set-of-values> is a list of values chosen between the following types : integer, real number, string between single quotes or <name-constant> .

<range-specification> is one of the following possibilities :

{ LOWER-THAN | LT
  GREATER-THAN | GT
  LOWER-EQUAL | LE
  GREATER-EQUAL | GE }

{ integer
  real-value
  string between
  single quotes }

{ EQUAL | EQ
  NON-EQUAL | NE }

{ integer
  real-number
  string between
  single quotes
  <name-constant> }

{ integer
  real-number
  string }

THRU

{ integer
  real-number
  string }

C. restrictions

- If 'THRU' option is chosen, the two values must possess the same type; the first one must be greater than the second one.

- If the range is defined as a ‹set-of-values›, the type of the values can be mixed. Those names must be separated by a comma.

D. Examples

```
RANGE R2 = 34 THRU 45 ;

RANGE R3 = 0.75 THRU 0.90 ;

RANGE R4 = LT 'book' ;

RANGE R5 = EQUAL HIGH ;

RANGE R6 = { HIGH,MIDDLE } ;
```

3.2.1.3. A Database Dedicated to the Query System

3.2.1.3.1. Description

In the existing version of the Query System, all the work carried out during a session is irremediably lost at the end of the session. This is one of the main reasons for the user non satisfaction. By introducing a database dedicated to the Query System (QS-DB), we meet this criticism.

If a user plans to introduce the same query during a further session, he may request the Query System to save the set in the QS-DB. At this moment, the following elements will be saved :

- The set definition entered by the user, nameley, its criterion.
- The parsed expression of the query : saving it will avoid a new interpretation of the query when this set is restored from the QS-DB. The parsed expression is also saved for a range.
- Possible comments about the set (see part 3.2.1.5).

During a further session, the saved sets and ranges can be restored in the current session and can be used just as other objects as they were defined during this session.

Selected object-names are not saved in the QS-DB even if they constitute a part of the set : we must keep the two databases (user-database and QS-DB) independent from each other because the user-database content can change during two QS-sessions. Therefore, the list of objects satisfying an indentical criterion could change between two QS-sessions. When the set is restored, it will be again evaluated basing on the current user-database.

A set may apply other sets or ranges within its definition. When a specific set is saved, all the objects it needs to be evaluated can be automatically saved with it ('RELATED' option in SAVE command).

Not all object modifications inside the QS-DB are allowed (as rename a set or change its definition). An object may only de destroyed in the aggregate. If the user wants to modify an object, he must first restore it in the current session (if useful), modify it in the current session, delete the old definition from the QS-DB and save the new one in the QS-DB.

Commands allowing to save objects (SAVE command), to restore objects (RESTORE command), to display their definitions (SHOW command), and to destroy or undestroy objects (DESTROY and UNDESTROY commands) are available and described below. Only the functional description is discussed. Technical specifications can be found in [IM47].

3.2.1.3.2. Commands Summary

3.2.1.3.2.1. DESTROY Command

A. Purpose

To erase and remove sets and ranges from the QS-DB. These sets and ranges are only logically destroyed, namely, the user can relocate them by using the 'UNDESTROY' command. The ranges and sets are physically destroyed at the end of the current session of the Query System.

B. Syntax

```
DESTROY <range-or-set-names-list> [RELATED]
```

C. Description

If "RELATED" is specified, all the sets which reference the listed sets will be destroyed, namely, all the sets that directly or indirectly need the listed objects to be evaluated are destroyed. A success message will be displayed for each object deleted from the QS-DB.

D. Example

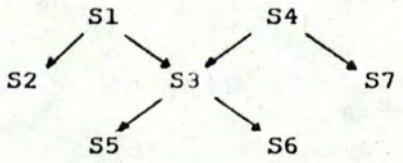
QS-DB contents

```
S1 = S2 OR S3 ;

S3 = S5 OR S6 ;

S4 = S3 OR S7 ;
```

The situation can be visualized as :



> DESTROY S5 RELATED ;
sets S5,S3,S1,S4 destroyed from the QS-DB.

> DESTROY S5 ;
set S5 destroyed from the QS-DB.

> DESTROY S1 RELATED ;
set S1 destroyed from the QS-DB.

3.2.1.3.2.2. Restore Command

A. Purpose

To restore the information about the saved sets and ranges from the QS-DB to the current QS session. Even though a QS-DB is attached for the session, the sets and ranges priviously saved in the QS-DB may not be referenced unless they are restored for the current session. After restoring, the sets and ranges still remain in the QS-DB.

Since no user-names list is saved in the QS-DB, the restored sets are evaluated basing on the current universe, namely, the current user-database if no universe is currently specified or the universe if it is currently defined.

B. Format

```
{RESTORE | REST} ( <set-and-range-names-list> | ALL )
                 [RELATED]
```

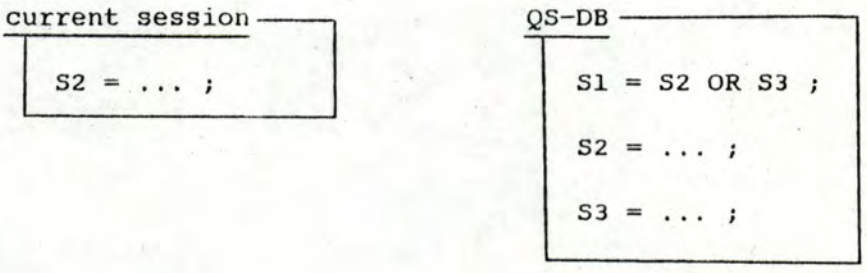
C. Description

If 'ALL' is used, all the sets and ranges which do not already exist in the current session are restored from the QS-DB to the current session.

If 'RELATED' is specified, all the sets and ranges which are referenced by the sets listed in the command and which do not already exist in the current session will be automatically restored from the QS-DB.

A set can be restored only if all the sets and ranges necessary for its evaluation already exist in the current session. If "RELATED" option is specified, the program itself will settle the restoring order. In the case where the user does it by himself, he must verify first that these objects already exist in the current session (by using the DISPLAY command in the current session and the SHOW command in the QS-DB).

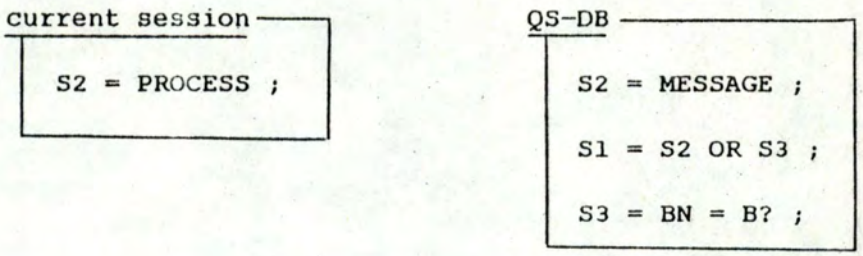
example :



It is not possible to restore S1 alone because S1 needs S2 and S3 to be evaluated; as S3 is not defined in the current session, the user must first restore S3 before restoring S1 or he may ask : "RESTORE S1 RELATED".

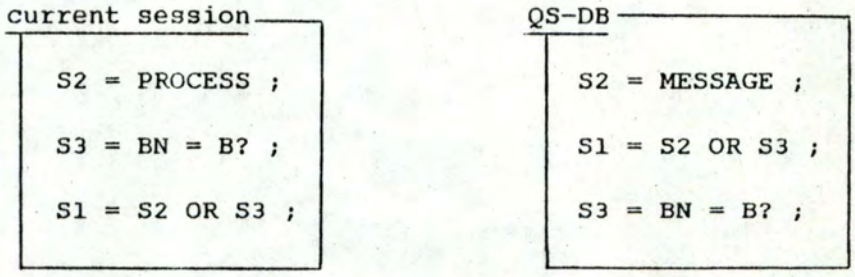
As sets and ranges names must be unique in the current session, two definitions of the same object cannot simultaneously exist in the current session. A clash might arise if the user wants to restore an object whose definition already exists in the current session. The basic priciple underlying this system behaviour is that for any definition of the set to be deleted, we must be able to retrieve it. Therefore, the existing definition is preserved in the current session and the definition contained in the QS-DB won't be restored. A warning will be displayed to point out this clash. However, if other sets reference the set under study, the existing definition will be used to evaluate these sets.

Example :




```
> RESTORE S1 RELATED ;
  warning - set S2 is currently defined in the
    current-DB; cannot be restored.
  set S3 restored in the current-DB.
  set S1 restored in the current-DB.
```

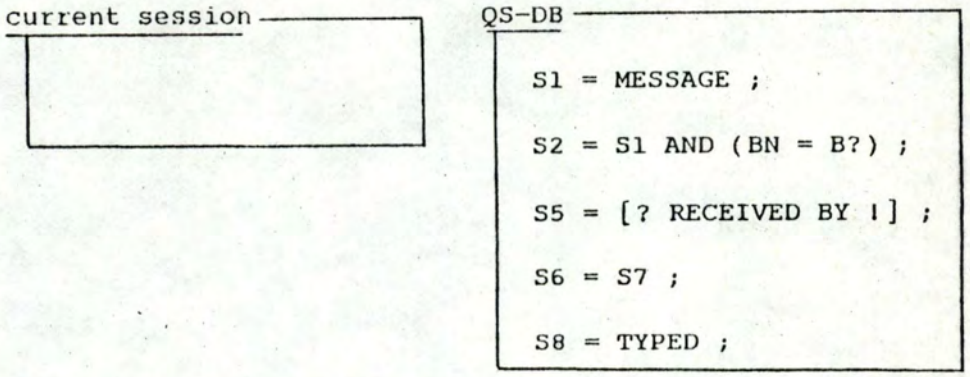
S1 will be evaluated with the existing S2 definition (S2 = PROCESS); the following situation will be found :



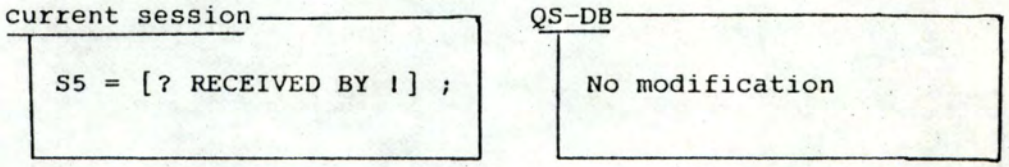
If a set definition contains the reference to a user-name that does not belong anymore to the user-database, the set won't be restored and with it, all the sets that need this set to be evaluated. A message will be displayed, mentioning the unappropriate user-name and the list of user-names that cannot be restored. The same case could also happen for constants in a range definition.

D. Examples

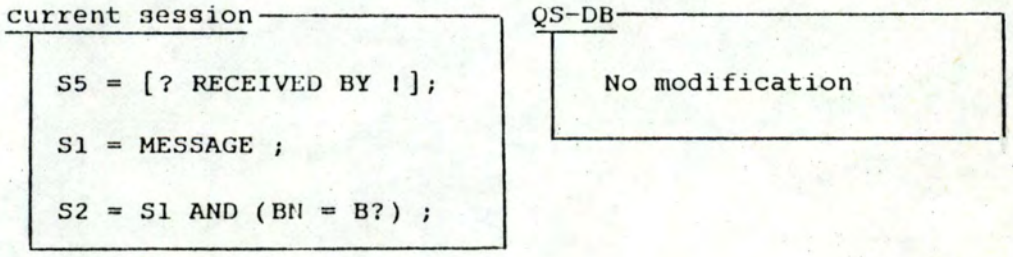
The chronogical sequence is as follows :



> RETORE S5 ;
set S5 restored in the current-DB.



> RESTORE S2 RELATED ;
sets S1,S2 restored in the current-DB.



```
› RESTORE S6,S8 RELATED ;
  warning - S6 needs definition of other
    objects - cannot be restored.
  set S8 restored in the current-DB.
```

current session

```
S5 = [? RECEIVED BY !] ;

S1 = MESSAGE ;

S2 = S1 AND (BN = B?) ;

S8 = TYPED ;
```

QS-DB

No modification

3.2.1.3.2.3. SAVE Command

A. Purpose

To save sets and ranges in a database dedicated to the Query System. After being saved in the QS-DB, the objects (sets and ranges) still remain in the current QS-session. Nothing is saved automatically in the QS-DB. Only objects saved in the QS-DB can be restored during a further QS session.

B. Syntax

```
SAVE ( ‹range-or-set-names-list› | ALL ) [RELATED]
```

C. Description

If "ALL" is specified, all the sets or ranges existing in the current session are saved in the QS-DB if they do not already exist in this database.

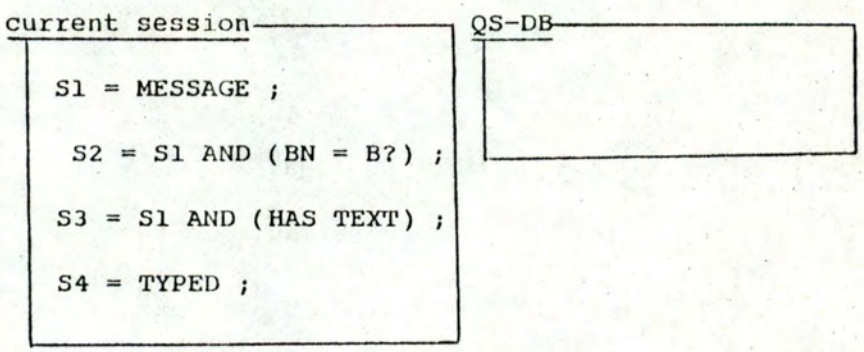
If "RELATED" is used, all the sets and ranges referenced by the sets listed in the command will be automatically saved.

The sets defined as a list of user-names cannot be saved in the QS-DB because their definitions are too dependent on the current situation of the user-database.

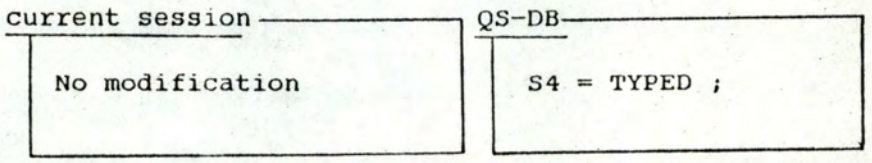
The set and range names are identifiers within the QS-DB. A warning message will be displayed if duplicates are introduced. If the user wants to change the object definition, he must first delete the old one from the QS-DB and then save the new one.

D. Examples

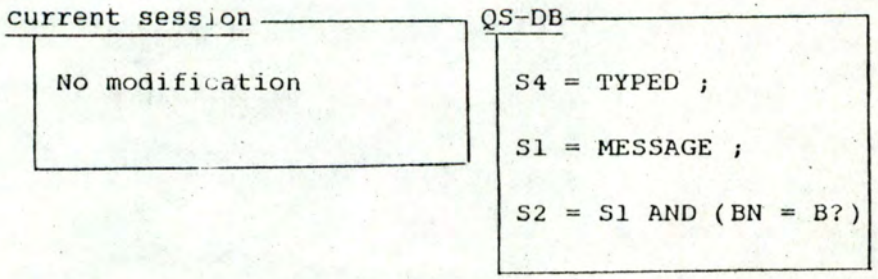
The chronological sequence is as follows :



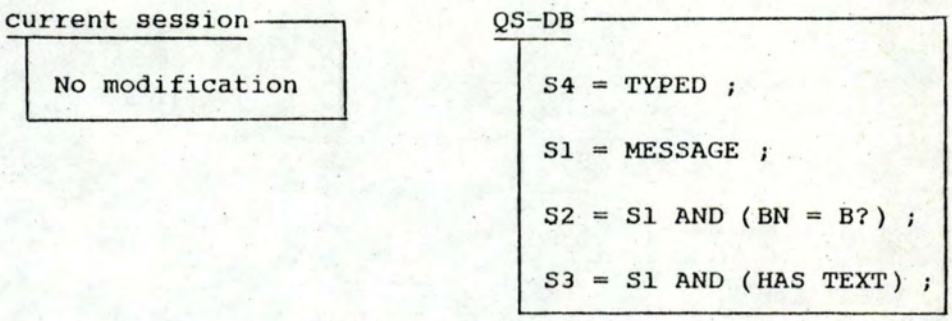
› SAVE S4 ;
set S4 saved in QS-DB.



› SAVE S2 RELATED ;
sets S1,S2 saved in QS-DB.



```
> SAVE S3 RELATED ;
warning - set S1 already exists in QS-DB -
          cannot be saved.
set S3 saved in QS-DB.
```

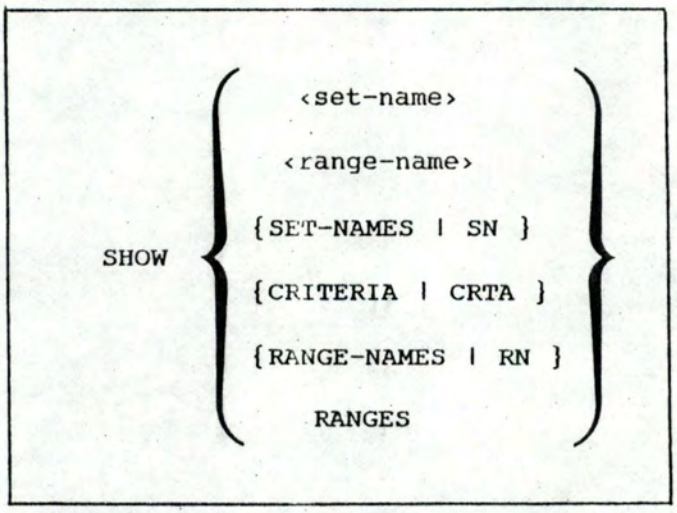


3.2.1.3.2.4. SHOW Command

A. Purpose

To display the definition of a set or of a range contained in the QS-DB, or to make the list of all the ranges-names,all the set-names,all the sets definitions or all the ranges definitions.

B. Syntax



C. Description

If a ‹set-name› is specified, the output is :
‹set-name› : ‹set-definition›.

If 'SET-NAMES' is specified, the output is the list of all set-names previously saved and not destroyed, classified by alphabetic order.

If 'CRITERIA' is specified, the output is the list of the set-names followed by their definitions; they are classified by alphabetic order.

If 'RANGE-NAMES' is introduced, the output is the list of all range-names currently defined in the QS-DB, classified by alphabetic order.

If 'RANGES' is specified, the output is the list of all range-names followed by their definitions; they are classified by alphabetic order.

D. Examples

```
SHOW set-1 ;

SHOW CRTA ;

SHOW RANGE-NAMES ;
```

3.2.1.3.2.5. UNDESTROY Command

A. Purpose

to retrieve sets or ranges previously removed (using DESTROY command) from the QS-DB during the current QS-session.

B. Syntax

```
{UNDESTROY | UNDES} ‹set-or-range-names-list› [RELATED]
```

C. Description

If "RELATED" is specified, all the sets that reference the objects listed in the command are also undestroyed and can be referenced again (the same mechanism as in DESTROY command).

If a new definition is entered between the DESTROY and the UNDESTROY commands, undestroying this set is not legal.

example :

If the following sequence is introduced :

> SET S1 = PROCESS ;

> SAVE S1 ;

> DESTROY S1 ;

> SET S1 = MESSAGE ;

> SAVE S1 ;

> UNDESTROY S1 ;
warning - set S1 was not previously destroyed.

D. Examples

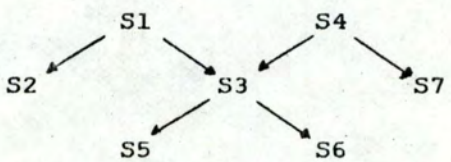
QS-DB

S1 = S2 or S3 ;

S3 = S5 or S6 ;

S4 = S3 or S7 ;

where all these sets were priviously destroyed.
The situation can be visualized as :



> UNDESTROY S2,S7 ;
sets S2,S7 undestroyed from QS-DB.

> UNDESTROY S5 RELATED ;
sets S5,S3,S1,S4 undestroyed from QS-DB.

3.2.1.4. Sets Evaluation From a Subset of the User-database

3.2.1.4.1. Description

In the current version of the Query System, queries are evaluated basing on the entire user-database, even if the user wants to control the extent to which the evaluation is performed. The 'SET UNIVERSE' command allows the user to specify the universe in which the queries will be evaluated, namely, a specific part of the user-database. The default universe is set to the entire user-database at the beginning of the session or reset during the session by using the

The definition of a universe is useful when many consecutive queries can be expressed on the same subset of the user-database. In this case, it is advisable to define a universe instead of conjuncting the universe definition with the set definition because the time needed to run the queries decreases substantially. However, this solution could introduce some confusion because when the user lists the set, he has to remember the universe in which the set was evaluated. The universe can be recalled by adding comments to the set.

A universe is defined using a criterion as in the definition of a set. the user may find back the specification of the current universe using the

We proceed to the description of the command which allows the definition of a subset of the user-database.

3.2.1.4.2. SET UNIVERSE Command

A. Purpose

To limit the subset in which the evaluation of the following queries are made.

B. Syntax

```
{SET UNIVERSE | SET UNIV} = ( DB | <criterion> ) ;
```

C. Description

If "DB" is specified, the universe is set to the entire user-database.

The criterion has the same definition as that used to describe a SET (see section 1.3.3). The criterion defining a universe is always evaluated basing on the entire user-database regardless of the current universe.

D. Example

```
> SET UNIVERSE = PROCESS ;
```

following queries will be evaluated basing on the subset of objects whose type is process.

```
> SET S1 = HAS PRIORITY HIGH ;

> SET S2 = [? TRIGGERED BY book-requisition] ;
```

these two queries only concern the process; therefore, there is no need to evaluate them basing on the entire user-database. They are semantically equivalent to the following queries :

```
> SET S1 = PROCESS AND HAS PRIORITY HIGH ;

> SET S2 = PROCESS AND [ ? TRIGGERED BY book-requisition ] ;
```

3.2.1.5. Comments Added to a Set

3.2.1.5.1. Presentation

A set can be defined as a hierarchy of other sets. Its definition can then become difficult to interprete. Moreover, the universe in which the sets are evaluated can change from one evaluation to the other. If a specific universe is defined, it is useful to remember it by explaining the set with some comments. These reasons induce us to allow the user to add some comment lines to a set.

These lines constitute a part of the set. They are displayed with its definition (DISPLAY command) or with the list of user-names (LIST command). The comments are also saved and restored with the set in the QS-DB.

Three commands allow respectively to add, to remove, or to modify comments of a set (respectively ADD COMMENT, DELETE COMMENT and REPLACE COMMENT commands). No comment may be added to a range. These commands may only be used with sets contained in the current session, but not with those contained in the QS-DB. Here follows their descriptions .

3.2.1.5.2. Comment Commands

3.2.1.5.2.1. ADD COMMENT Command

A. Purpose

To add textual comment lines to the named set.

B. Syntax

```
ADD  {COMMENT | CMT}  [FOR]  <set-name> ;
```

3.2.1.5.2.2. DELETE COMMENT Command

A. Purpose

textual comments existing for the named set are deleted.

B. Syntax

```
{DELETE | DEL}  {COMMENT | CMT}  [FROM]  <set-name> ;
```

3.2.1.5.2.3. REPLACE COMMENT Command

A. Purpose

To replace the current textual comments for the named set with new ones.

B. Syntax

```
{REPLACE | REPL}  {COMMENT | CMT}  [FOR]  <set-name> ;
```

3.2.1.6. Rules Introduced in a Basic Criterion Construction

The existing QS version allowed a user to define an intricate query by specifying only one expression. By using the nested queries, the user can avoid the splitting of a query into a hierarchy of other queries easier to interprete. Using such a decomposition methodology is justified in case of intricate requests even if this mechanism reinforces the procedural aspect of the language.

To facilitate user interpretation of the expression, we suggest the adoption of a step-by-step methodology in defining queries by removing the possibility to imbricate queries. Therefore, it won't be possible to introduce a basic criterion inside a statement. The basic criterion will be defined first as an independent set; the name of the created set will then be specified in the statement instead of the basic criterion.

Example :

In the existing version, the query "select messages which are generated by process which possess no description" is expressed as :

```
SET  S1 = NOT HAS DESCRIPTION GENERATES ! TIMES ? IF ! ;
```

In the new version, it will be expressed as :

```
SET S1 = NOT (HAS DESCRIPTION) ;

SET S2 = [ S1 GENERATES ! TIMES ? IF ! ] ;
```

The object designation occuring in a statement are now chosen in the following list :

- user-name : book, reader, ...
- set-name : S1, SET-1, ...
- "!" to designate 'anything'
- "?" to designate what must be selected
- range-name when it makes sense : a constant is supposed to be used at this place in the statement : R1, RANGE-1, ...

3.2.1.7. Miscellaneous Commands

Two other commands have been added to facilitate the set and the range manipulation. They are described below and are designed to complete the facilities offered by the system to respond to issues relative to users non satisfaction.

3.2.1.7.1. ERASE Command

A. Purpose

To remove set or range names and associated data from the current session, not from the QS-DB. The sets and ranges may not be referenced during the current session; they may be restored if they were saved before in the QS-DB (by using the RESTORE command).

B. Syntax

```
ERASE  <list-of-objects> [RELATED]  ;
```

C. Description

If "RELATED" is specified, all the sets which reference the erased sets and ranges will be also destroyed (a range cannot reference another range).

D. Example

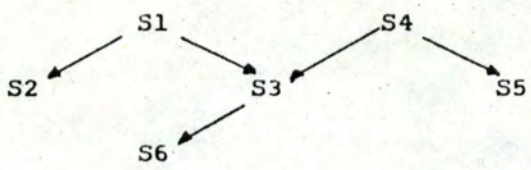
Current session

```
S1 = S2 OR S3 ;

S3 = S6 ;

S4 = S3 OR S5 ;
```

the situation can be visualized as :



> ERASE S5 ;
set S5 erased from the current-DB.

> ERASE S6 RELATED ;
sets S6,S3,S1,S4 erased from the current-DB.

3.2.1.7.2. EXECUTE Command

A. Purpose

To execute queries or other commands contained in a file whose name is specified in the command. The program executes those commands from the first one until the last one. An echo will display these commands on the screen; commands with syntactical or semantical errors will be ignored.

B. Syntax

{EXECUTE | EXEC } [FROM] <file-name> ;

3.2.2. Implementation Level

3.2.2.1. Introduction

This section will restrict itself to present the main differences between the existing QS version and the new one from an implementation view point. A more complete desription of the new version implementation is given in chapter 4.

The new QS structure is shown in Figure 3.1 . Two autonomeous analyzers are integrated in the Query System. The first one, call LANG-PAK, performs the lexical analysis of the statements expressed in the Query Language. The second one, SNTX, achieves the analysis of the statements expressed in the target language. These two aspects of the implementation are described below. A third one is also discussed; it concerns the internal data structure.

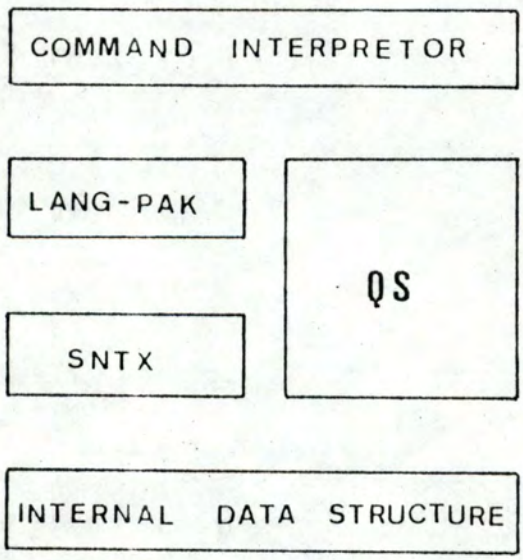


Figure 3.1 : QS Program Structure.

3.2.2.2. Using LANG-PAK to Describe and Use the Query Language

3.2.2.2.1. Introduction

LANG-PAK is an interactive language design system which aids a language designer in designing and implementing simple interactive computing languages. By using this software, we achieve the two phases of a language definition : LANG-PAK allows a "fast" and "comfortable" definition of a simple command language and facilitates any further language modification [HEIN.75] . Comfortable construction because no program needs to be written to perform the lexical analysis of the described language; ease of modification because the language design methodology allows the redefinition of a part of the language whatever is the system application step we are performing. Moreover, using the LANG-PAK software allows the designer to achieve the target language lexical analyzer independently of the rest of the QS-program.

We are not going to insist on the tool portability aspect and ease-of-use possibilities. Let us point out that a version in FORTRAN is provided in [HEIN.75] . Its installation requires only few hours to program the system-dependent subroutines.

We describe LANG-PAK by following the different steps in defining and using a command language. Afterwards, we evaluate this software and point out its limitations and the modifications introduced to the basic version in order to implement our Query Language.

3.2.2.2.2. The Language Application Phases

The design of a language using LANG-PAK is devided in two phases : the language definition and its practice. These two steps are realized according to an identical principle in LANG-PAK : both of them are supported by a meta-language, and the same program is used to realize these two phases (Figure 3.1).

3.2.2.2.2.1. The Language Definition Phase

At first, we need to define the grammar of the target language, namely, the command language we want to describe. During this phase, we introduce the language definition using the LANG-PAK meta-language.The final purpose of this phase is to generate the tables containing the grammar. These tables are called the grammar lexicon.

The definition is performed using the meta-language. The meta-language provided in this software is highly similar to the BNF notation. The main difference with the BNF notation is due to the fact that the left recursivity is removed by introducing the limitated repetition concept. Table 3.3 contains the meta-language elements used in the Query language definition.

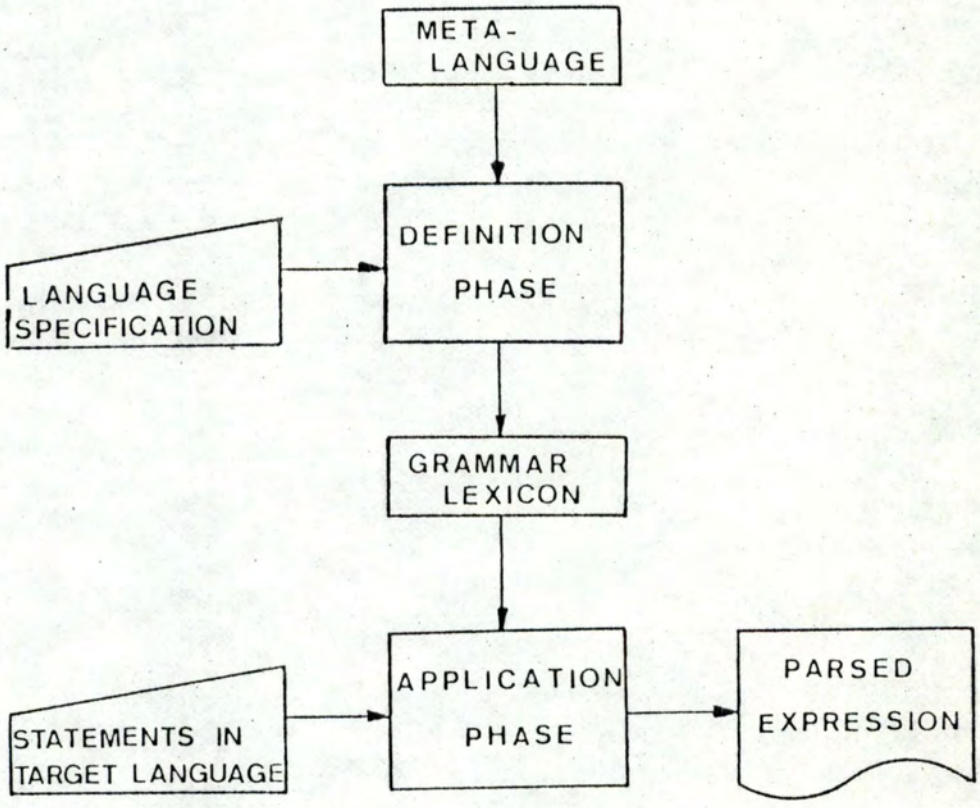


Figure 3.2 : Using LANG-PAK in a language application

Meta language element	definition of element
"...." (..)	A non-null terminal symbol sequence enclosed in quotes called a literal whose integer attributes are contained in parentheses.
I	Any member of the terminal word set integer.
N	Any member of the terminal word set number.
V	Any member of a language designer defined set of terminal words.
S(..)	Any member of the terminal word set string which is followed in the input symbol sequence by a member of one of the terminationbreak sets.
EOS	Parse machine instruction to test for end of symbol sequence.
(...) (i j)	A repeating specification to be iterated at least i times but not more than j where (j >= i >= 0) and (j > 0)
!	exclusive OR
&	conjunctive AND
<.....>	a language specification type

Table 3.3 : Elements of the Meta-language used in the QL definition

In the Query Language definition, most of the language reserved words are defined as "literals". They are isolated by the LANG-PAK parser and translated into internal code. It is the case for :

- command names (SET,HELP,SAVE,..)
- reserved words in the basic criterion :
  (BN=,TYPED,PROPERTY,..)

- options in commands (PRESENTATION,BASIC,..)

- logical operators (NOT,AND,OR)

Object-type names, user-names,set-names and range-names are defined as "string" and are analyzed in the QS-program itself because they reference respectively the meta-database, the user-database and the current session.

Example :

The SAVE command whose syntax is :

```
SAVE name[,name]  [RELATED] ;
```

is defined as :

```
<save-cmd> = "SAVE"(100) <name-list> ( "RELATED"(10) ) (0 1) ;

<name-list> = ( <first-name> ) (0 9) <last-name> ;

<first-name> = S(",")  "," ;

<last-name> = S ( "RELATED" V ) ;
```

- The reserved words are defined as literals : 'SAVE' and 'RELATED'. Their attributes are respectively 100 and 10

- Set-names are defined as strings. We need to define in a different way the first names and the last one because their ends of strings are specific.

The entire definition of the Query Language in the LANG-PAK meta-definition can be found in Annex-2.

In order to help the language adjustment, LANG-PAK provides the user with some tools and possibilities. As an illustration, the user may test the definition just entered by using some examples : since each block of language specification types (LST) is autonomeous, the user can test a part of the language without having to introduce the entire definition. For instance, we could already enter a statement using SAVE command without finishing the language definition; however, we need to introduce the four LST given in the example to test the SAVE command because they all are used in its definition and constitute a LST block. Another tool provided in LANG-PAK is called the "TRACE" command. It allows to follow the LANG-PAK parser when it analyzes a statement and facilitates the error location in the definition.

At the end of this phase, the grammar lexicon is saved into an external file. This file is used to communicate between the two phases. With this method, any target language modification is easily performed : it is "sufficient" to introduce another definition and save it in the file.

3.2.2.2.2.2. The Language Use Phase

During the second phase, we use the target language itself : statements expressed in target language are parsed. after reading the grammar lexicon from a file, the parser will determine if a statement is syntactically consistent. In this case, it will produce a parsed expression. Table 3.4 contains meta-language terminal elements translation into internal code.

Type	Meta-language representation	translation code	Arguments
Integer	I	1	value of integer
Real number	N	2	value of real number
String	S(..)	3	length of string symbol sequence in EBCDIC code
Literal	"..."(..)	5	Number of attributes List of attributes

Table 3.4 : Meta-language terminal elements translation

The QS-program itself has then to use the internal code to understand the command introduced by the user.

Example :

Following the SAVE command definition, the statement :

" SAVE S1,R1 RELATED "

will be translated as :

token	type	translation
SAVE	literal	5 1 100
S1	string	3 2 226 241
,	literal	null translation
R1	string	3 2 217 241
RELATED	literal	5 1 40

The same program realizes the second phase. However, tools whose purpose is to help the language design can be removed. This subroutine subset is integrated in the QS-program itself.

3.2.2.2.3. The Dark Side of LANG-PAK

This software involves limits and defects in its design. Some improvements are proposed in this chapter. On is already realized in the version we implemented.

The class of languages which can be described using LANG-PAK constitutes a first limitation of this software : the target language must be simple, namely, it must possess a restricted syntax and a few production rules. A language like the Query Language is already too complex. Even though the Query System seems to be easy to describe, we were forced to add to the language some additional signs which add nothing to the language semantic : they are the square brackets "[..]" to delimit the statements and the braces "{..}" to specify a set of user-names.

We mentioned the independence between the syntactical analyzer and the QS-program itself as one of the greatest benefit in using LANG-PAK. However, there exist some limitations to this independence : the interface, namely, the parsed expression must keep the same structure whatever the form of the language. Any further modification of the target language will be achieved within these limits.

Some expressions cannot be expressed with the meta-language because in the list of break characters for a string, the meta-language introduces an evaluation order, even if this order is not significant. A new lexical analyzer should be implemented, based on the TOKEN concept, i.e., a string ending by any character contained in a predefined list.

During an exclusive enumeration of literals, LANG-PAK also introduces an evaluation order, even if the user wants that the more suitable literal is chosen.

Example :

Following the definition :

```
<a> = ( "DISP"(110) | "DISPLAY"(110) ) S ;
```

the input command "DISPLAY SET-1" will be decomposed as :

```
DISP        :  literal   :  attribute = 110
LAY SET-1   :  string    :  length = 9
```

even though the result is supposed to be :

```
DISPLAY     :  literal   :  attribute = 110
SET-1       :  string    :  length = 5
```

When a statement is introduced, the basic LANG-PAK version allowed the user to devide a command in many lines by using the continuation sign "-". This version has been modified because it seems us to be more suitable in the Query Language to continue the introduction until a break-sign is introduced. Henceforth, the user will define in the beginning of the LANG-PAK session the break-signs set. Any further introduction will continue until one of these break-signs is matched.

3.2.2.3. Introducing SNTX to Parse a Target Language Statement

A part of the Query System syntactical and semantical analyzer performs the analysis of the statements expressed in target language. Since there was no general parser for the target language statements when the initial version was implemented, the parser of the Query System performs all the above-mentioned parsing. Therefore, the structure of the Query System becomes very complex. Since the general parser, called SNTX, is now available in the ISDOS project, the Query System should utilize this parser for the parsing of description language statement. By utilizing the SNTX parser, the structure of the Query System will become much more simple and the target language parsing scheme may be standardized throughout ISDOS software.

SNTX cannot be integrated in the Query System without some modifications, since this analyzer was designed to parse Input Processor statements. Here follows the list of modifications introduced in the form of an interface to integrate SNTX in the Query System.

The final purpose of these two parsers (in the IP and in the QS) are basically different, even if the parsing itself is similar. In the IP, new statements are introduced in the user-database after having checked that these statements are syntacticaly and semanticaly constistant. In the Query System, the same checkings are performed but here, to select from the user-database objects which are involved in the relationship expressed in the statement. This part of SNTX has to be modified.

The list of elements occuring in a statement are specific : in the IP, elements are either user-names or list of user-names. In the QS, they are :

- the question mark
- the exclamation point
- the user-name
- the set-name
- the range-name

We removed the possibility to specify object-type names as element in the Query System because an identical name can be used as object-type and keyword; therefore, ambiguity is introduced in SNTX.

Example

The two following statements belong to the target language :

(1) SUBPARTS PROCESS [ARE] process-name ;

(2) SUBPARTS [ARE] name ;

If the query is expressed as " ? SUBPARTS PROCESS ", the SNTX parser cannot identify the object-type because the same name is used as keyword in the statement (1).

This ambiguity restricts the Query Language possibilities. The query expressed in the example can still be stated but by constructing first another set containing all the objects whose type is "PROCESS".

example :

The query " find all the messages received by any interface " was expressed in the old version as :

```
SET S1 = ? RECEIVED BY INTERFACE ;
```

In the new version, it will be expressed as :

```
SET S1 = INTERFACE ;

SET S2 = [ ? RECEIVED BY S1 ] ;
```

Every statement in the IP references to a block-header. In the QS, this block-header is replaced by the first element (front-end corner).

example

In the Input Processor, we find :

```
DEFINE PROCESS P1 ;

    ON TERMINATION TRIGGERS P2 ;
```

3.2.2.4. Isolating the Internal Data Structure Enhances QS Modifiability

In programs like the QS, any answer to a question from the user requires at least one access to the Internal Data Structure (IDS), namely , the structure containing informations about created sets and ranges cluring a QS session. It would be desastrous if any IDS modification had consequential effects in the entire program, whatever the reason for introducing this modification (a new concept introduction, performances increasing,...).

This fact leads us to isolate the IDS from the rest of the program. In the new QS version, a set of subroutines will constitute the interface between the QS and the IDS. In this way, any IDS modification will induce changes only within the interface.

The IDS interface is built as a set of low-level subroutines manipulating the basic notions (set and range). As an illustration, we mention the subroutines whose purpose is :

- to create a set
- to get the set definition
- to delete a set
- to add comments to a set
- to check if a set is defined

The same subroutines also exist to manipulate ranges.

3.3. NEW QUERY SYSTEM VERSION EVALUATION

3.3.1. Functional level

References between square brackets refer to criteria defined in chapter 2.

3.3.1.1. Language Intrinsic Qualities [C1] - [C2]

The following remarks and criticisms can be formulated with respect to the new version :

1. The expression of the criterion is the same as in the existing version.

2. the criterion notion is removed and with it, the ambiguity it introduced to the user. The CHECK command does not make sense anymore without the criterion concept. However, it can be achieved by using the SET notion :

   Example :

   To check if the user-names list L satisfies a criterion C, the following sequence is introduced.

   ```
   SET S1 = L ;

   SET S2 = C ;

   SET S3 = S1 AND S2 ;
   ```

   S3 will contain user-names which satisfy the criterion C.

   ```
   SET S4 = S1 AND NOT (S2) ;
   ```

   S4 will contain user-names which do not satisfy the criterion C.

CHECK command can be removed without the lost of functionnality.

3. The RANGE concept manipulation and its usage is similar to these for the SET concept. The insertion in a statement is achieved in the same way.

4. Some command names have been modified because the new name is more significant :

EXPLAIN becomes HELP

CHANGE becomes RENAME

5. Special characters have been added even if they have no semantical justification; they are introduced for the implementation reasons (square brackets to delimitate a statement and braces to delimitate a user-names list).

6. We decided to duplicate some command names if they are used in the current session or in the QS-DB instead of introducing a new parameter . The experience has shown that it is easier to remember those names then to lengthen the command. It is the case for :

QS-DB		current QS session
SHOW	----->	DISPLAY
DESTROY	----->	ERASE

To display the universe, we used the DISPLAY command. Introducing a new command was not justified in this case.

3.3.1.2. Functions Offered By The Query System [C3]

1. The number of commands offered in the Query System has been doubled. They respond to a user need. However, a command which allows to classify queries is still missing and should be implemented to increase the facilities proposed by the Query System.

2. SET UNIVERSE command has been introduced to improve the execution time. However, this command constitutes no direct answer to the problem and it can introduced some confusion to the user : Indeed, the user needs to remember the universe in which his queries were evaluated.

3.3.1.3. System Usage [C4] -> [C7]

The execution time is still the aspect that needs to be improved. Introducing SNTX as analyzer for a statement slightly decreased the execution time but not in a significant way. Some proposals will be formulated in chapter 4 to improve this aspect.

3.3.2. Implementation Level [C8] -> [C10]

Problems pointed out in [C8], [C9] and [C10] have been solved. The new Query System structure can be visualized in Figure 3.1 .

CHAPTER 4 :

IMPLEMENTING

THE QUERY SYSTEM

4.1. INTRODUCTION

Few books and articles give an overview of the implementation aspect in describing a database query system. This chapter restricts itself to outline the implementation level of the Query System. It seems to us that discussing this aspect is useful to complete the Query System description.

The Query System general approach will be illustrated through an analogy between the Query System described in this thesis and an operating system. Both of them comprise some components which are relatively autonomous : as an operating system comprises a compiler, a linker, an editor, etc., we can isolate from the Query System specific components. However, as a program in source language must undergo a sequence of transformations in order to be executed (compile, link,save and run), a query expressed in the Query Language will successively pass through specific steps to be interpreted.

We begin this chapter by drawing the steps in analyzing a query and their chain. Then, we analyze in detail each of these steps by criticizing if necessary their current implementation. Finally, we present a detailed description of a last component of the Query System, the QS-database manager by describing the database structure. We will conclude this chapter by reviewing two aspects of the QS realization.

4.2. THE COMPONENT CHAIN IN INTERPRATING A QUERY

A query expressed in the Query Language must undergo three steps in order to be interpreted. Figure 4.1 describes the chronological trend of events.

During a first step, the query will be matched against the syntactic and semantic rules describing the query language. The query is decomposed by isolating the language keywords : This series of keywords will be checked basing on the language definition. Moreover, some arguments in the query must be analyzed depending on the context in which the query is stated : sets and ranges defined so far and current state of the user-database. The semantic analysis will take this context into account.

On recipt of an expression in which each basic criterion is syntacticaly and semanticaly consistent, the synthesizer will check if the combination of basic criteria makes sense (parenthesis rules). It then transforms the expression into postfixed Polish notation.

Finally, the set of objects satisfying the criterion will be selected from the current universe (entire user-database or subset of this database if a universe is currently defined).

Errors can be detected during the two first steps. In this case, an error message is displayed on the screen and the query analysis is interrupted.

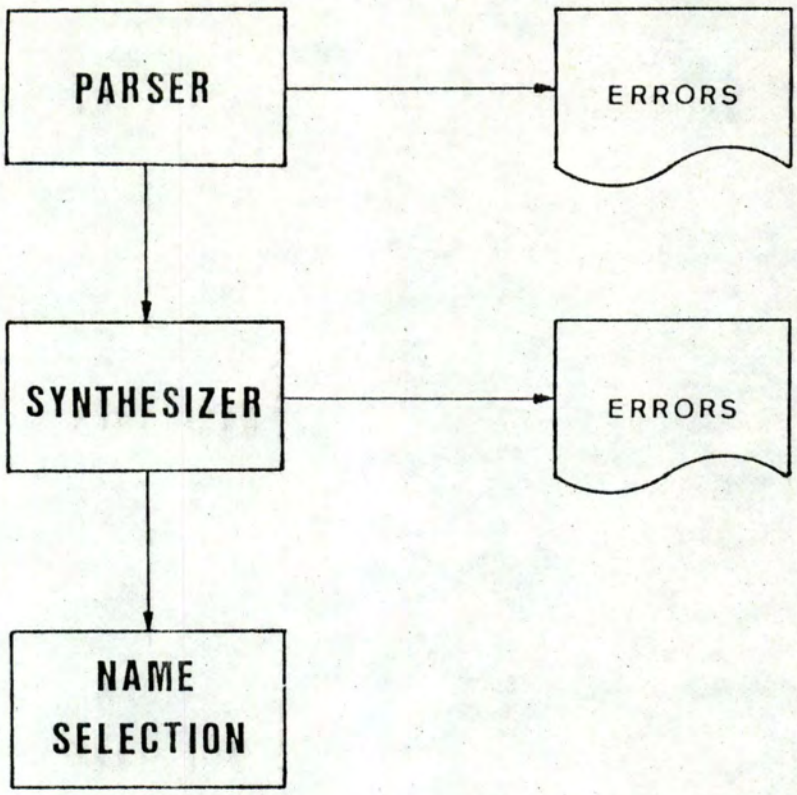


Figure 4.1

4.3. THE COMPONENTS

4.3.1. The Parser

The purpose of this component is to check if a query expressed using the Query Language is syntacticaly and semanticaly consistent. These two checks are performed sequentialy. This cutting out is due to the use of an external software to perform the syntactic analysis.

On recipt of the query as expressed by the user, the parser isolates the keywords from the query and checks its consistency with regards to the Query Language definition. These checks are achieved by the LANG-PAK software, described in part 3.2.2.1. As specified before, the LANG-PAK analyzer isolates the following elements :

- the command name (SET,LIST,RENAME,...)
- reserved keywords (BN=, TYPED,PROPERTY,...)
- logical operators (NOT,AND,OR)
- parenthesis.

other arguments are isolated as "strings" and transmitted to the semantic analyzer for further checks. The statements of the meta-language are also isolated as "strings" and analyzed by calling SNTX (see part 3.2.2.2).

In case of syntactic error, a message is displayed and any further analysis of this query is cancelled. The message displayed on the screen has always the same format, whatever is the syntactic error. More explicit informations could be provided to the user about the error type but in order to realize this, the error must be intercepted inside LANG-PAK and this is rather difficult to achieve.

Finally, the query is parsed with regards to its context, namely, the user-database and the sets and ranges currently defined. Any element not isolated as keyword during the syntactic analysis is checked by the semantic part of the parser. Examples are presented below; they describe semantic checks performed from a query :

Example :

```
> SET set-1 = (DLC <= 07/17/1983) AND [? GENERATED BY pro-1];
```

set-1 : this name may not be already assigned to a set or a range currently defined.

07/17/1983 : 1 <= month <= 12
1 <= day <= 31

? GENERATED BY pro-1 :

- one and only one question mark used in each statement
- the degree of the relationship (here : 2) must be in the ranges specified in the meta-language
- 'pro-1' must reference to an object in the user-database

The parsing of a statement is achieved by SNTX. Further checks must be performed because the parts in a statement may have specific values when they are used in the Query Language (set-name, range-name, '?', '!').

Each semantic and semantic inconsistency is pointed out to the user : the inappropriate part is displayed and the reason of its inconsistency is explained. Every command syntacticaly or semanticaly inconsistent is ignored.

4.3.2. the Synthesizer

On recipt of the definition of a new set judged syntacticaly and semanticaly consistent, the synthesizer translates the query expression in postfixed Polish notation. It should be noted that, in this language, the logical operators are : AND (intersection), OR (union) and NOT (complement). Parenthesis may be used to clarify the expression or to modify the operators priority rules.

Example :

S = ((C1 AND C2) OR C3) AND (C4 OR C5)

Where Ci are basic criteria, S will be translated as :

S = C1 C2 AND C3 OR C4 C5 OR AND

Two basic rules are followed :

1. All operators have the same priority; if no parenthesis are used, the expression is evaluated from left to right.
2. If no parenthesis are used, NOT-operator refers to the nearest argument.

One error can be detected at this level. It concerns the parenthesis : the number of the left parenthesis must equal the number of right parenthesis. The final query expression is produced as output of this step : the parsed stack. It will be used to select objects from the user-database during the name-selection.

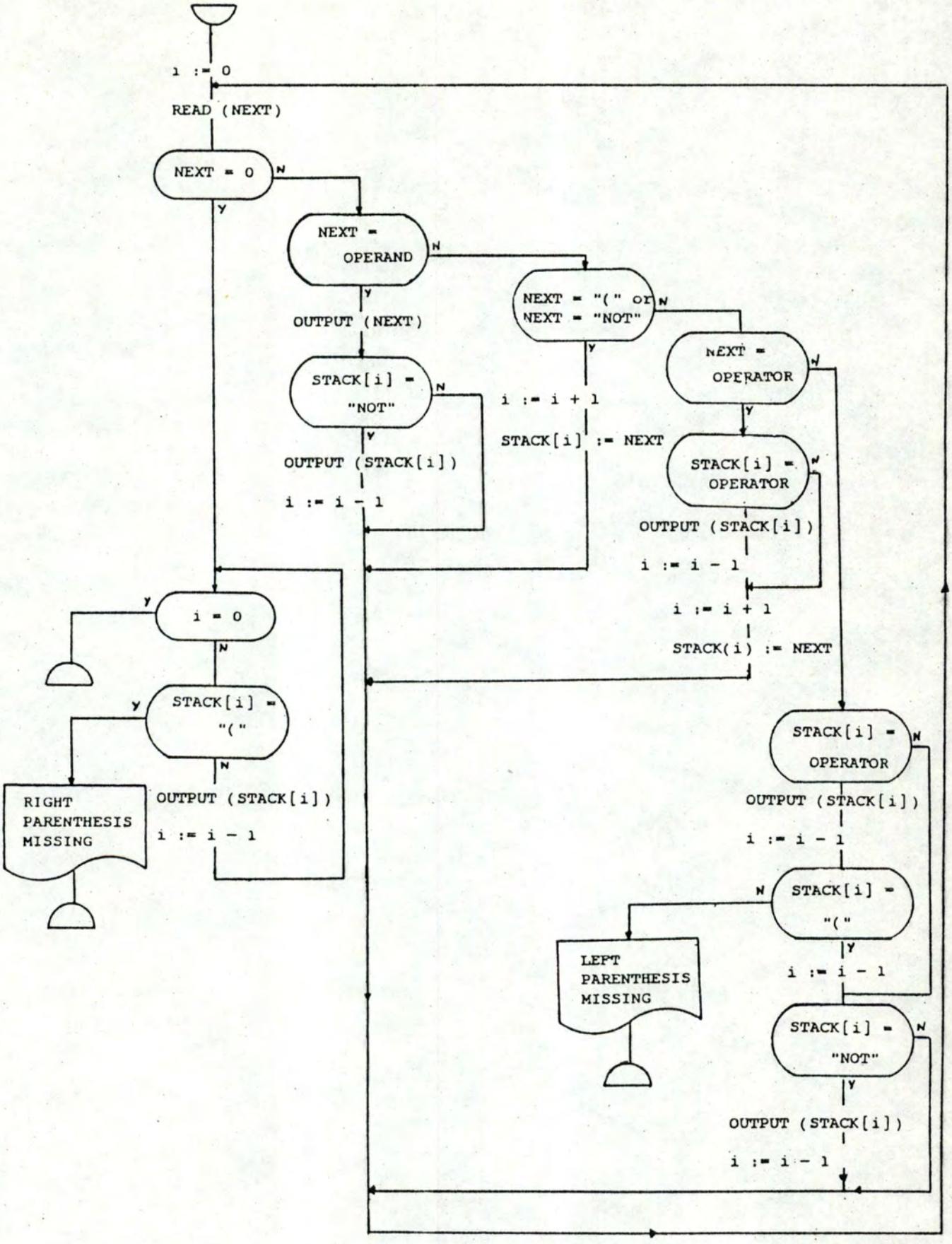

i := 0
READ (NEXT)
NEXT = 0
N
Y
NEXT = OPERAND
N
Y
OUTPUT (NEXT)
STACK[i] = "NOT"
N
Y
OUTPUT (STACK[i])
i := i - 1
NEXT = "(" or NEXT = "NOT"
N
Y
i := i + 1
STACK[i] := NEXT
NEXT = OPERATOR
N
Y
STACK[i] = OPERATOR
N
OUTPUT (STACK[i])
i := i - 1
i := i + 1
STACK(i) := NEXT
i = 0
Y
N
STACK[i] = "("
Y
N
RIGHT PARENTHESIS MISSING
OUTPUT (STACK[i])
i := i - 1
STACK[i] = OPERATOR
N
OUTPUT (STACK[i])
i := i - 1
STACK[i] = "("
N
Y
LEFT PARENTHESIS MISSING
i := i - 1
STACK[i] = "NOT"
N
Y
OUTPUT (STACK[i])
i := i - 1

Figure 4.2

4.3.3. The name selection

A. The current algorithm

Receiving the parsed stack, this component of the Query System selects objects satisfying the criterion described in the parsed stack from the current universe (entire user-database or specific universe if currently defined). In the existing version, one checks if each object satisfies the criterion, namely, we go over all the parsed stack for each object of the universe.

example :

the set S1 is defined as :

S1 = (C1 AND C2) OR C3

where Ci are basic criteria as defined in part 1.3.3. S1 will be translated by the synthetizer as :

S1 = C1 C2 AND C3 OR

For each object Ai belonging to the universe, one will check if it satisfies C1, then C2; then execute the logical 'AND', evaluate C1 and execute the logical 'OR' to get the final answer.

B. Criticism of this Implementation

Some authors have already analyzed the problem of selection optimisation. Indeed, they have rightly understood that the selection has a main impact on the response time. The Query System described in this thesis is also concerned by this problem : its performance is still badly decreased because of the name selection implementation. For its defense, we can mention the following fact : every proposed optimisation we found in the litterature to improve the response time ([ASTR.75], [KIM.82] or [YAO.79]) requires a database organization allowing direct access to data giving many access keys. This possibility is missing in the existing situation of the project.

C. Proposals to Improve the Name Selection.

Based on these observations, two new evaluation methods are proposed. The first one is based on the existing possibilities to access the user-database, the other one assumes a multiple-keys access.

These two solutions are based on a binary tree built from the criterion : its leaves represent the basic criteria, the nodes are the boolean operators

Example :

S = ((C1 AND C2) OR C3) AND (C4 OR C5)

where Ci are basic criteria. The corresponding binary tree is :

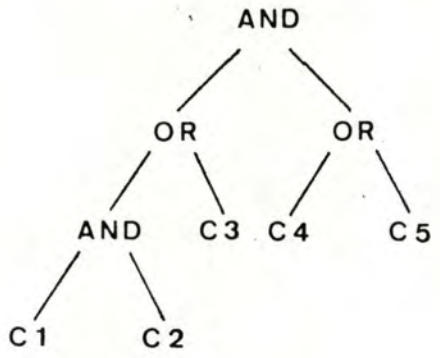


Figure 4.3

At this level, two solutions are considered :

1. we cannot avoid to scan entirely the universe in the existing circumstances. However, we can perceptibly reduce the evaluation time by using the binary tree and the properties of the logical 'AND' and 'OR' : when the expression is 'C1 AND C2', if C1 is false, the entire expression will have the same result in any cases. In the same way, for 'C1 OR C2', if C1 is true, the expression will always be true. In these cases, the evaluation of the second basic criterion is redundant.

   The algorithm can still be improved by ordering the basic criteria depending on their supposed evaluation time : indeed, it is more time-consuming to evaluate ' ? RECEIVES ! ' than 'MESSAGE'. By ordering the basic criteria, the less time-consuming ones are evaluated first and if some criteria evaluation is not required, the evaluation time will decrease.

2. The second solution seems to be faster : to evaluate separatly each leaf of the tree, namely, select objects satisfying the basic criteria corresponding to the leaves. Afterwards, the program executes the intersection or the union of these subsets by proceeding from the leaves towards the root.

example :

following the situation depicted in Figure 4.3, we obtain the following sequence of instructions :

evaluate C1 --> get list L1

evaluate C2 --> L2

L1 AND L2 --> L3

evaluate C3 --> L4

L3 OR L4 --> L5

evaluate C4 --> L6

evaluate C5 --> L7

L6 OR L7 --> L8

L5 AND L8 --> L9

This method also permits us to avoid to scan entirely the universe by only selecting objects in accordance with the basic criteria. However, this method is inadaptable in the existing situation : the access method to the user-database does not allow the selection of objects on the basis of their dates-of-last-change or a part of their basic names.

4.4. THE QS-DATABASE MANAGER

The purpose of the QS-database manager is to interface the QS-database with the QS program itself. Let us analyze the database structure and see how the QS-database manager uses this structure to perform its functions.

4.4.1. QS-database General Structure

The QS-database is implemented using ABDMS version D3.2 .ABDMS is a transportable general-purpose database management system based upon the CODASYL 1971 DBTG model for the network database approach, as extended or restrected due to user requirements and ISDOS experience with ABDMS operation. More informations on this DBMS can be found in [WP191].

Figure 4.4 gives the general structure of the QS-database. Object types are depicted by a rectangular box. Their names are located in the boxes. Access paths are represented by an arrow, joining the object types owner and member. Their names and connectivities are also mentioned. The structure description in DBMS/DDL can be found in Appendix-3.

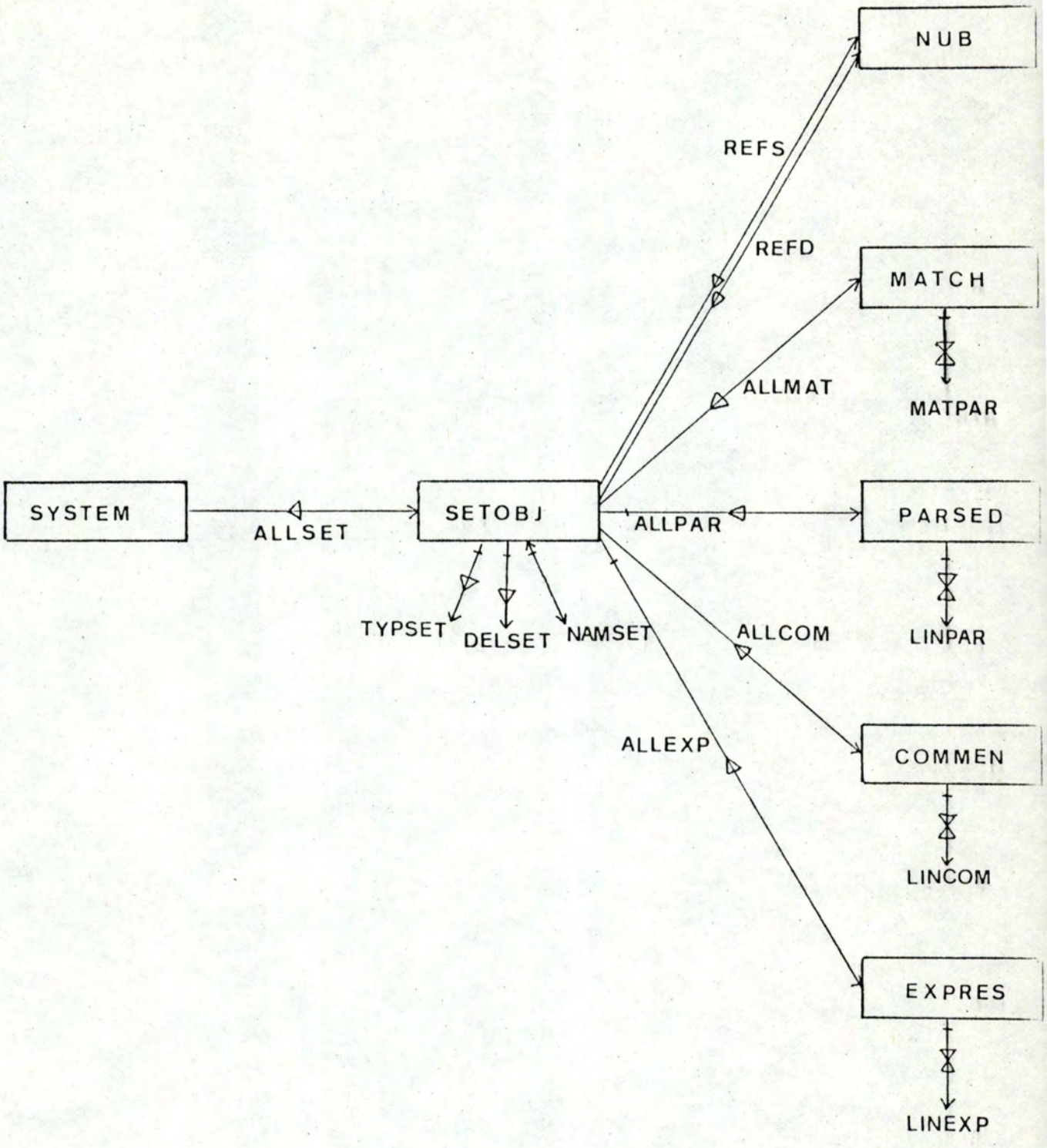

NUB
REFS
REFD
MATCH
ALLMAT
MATPAR
SYSTEM
ALLSET
SETOBJ
ALLPAR
PARSED
TYPSET
DELSET
NAMSET
ALLCOM
LINPAR
ALLEXP
COMMEN
LINCOM
EXPRES
LINEXP

Figure 4.4

4.4.2. The Records

SETOBJ

This record represents either the logical entity "SET" or the logical entity "RANGE" defined in the Query System. Each object is identified by its name (item NAMSET). Two other items are associated to the SETOBJ : its type, i.e., set or range (item TYPSET) and a "destroy" indicator (item DELSET) : it allows the user to check if the object was destroyed during the current session. Those objects are only physically destroyed at the end of the session.

EXPRES

An expression is associated to each set and range, namely, the definition as it was specified by the user. The expression is decomposed into several lines (each line has at most 80 characters). An occurence of the EXPRES record-type corresponds to each line.

COMMEN

Comments can be associated to a set. An occurence of the COMMEN record-type contains a comment line (at most 80 characters). Many lines can be associated to a set. A range may not possess any comment.

PARSED

A parsed stack is associated to each set and range (see 4.3.2). Each element from the stack is an integer and corresponds to an occurence of the PARSED record-type.

MATCH

The set or range definition could possess a reference to objects contained in the user-database : indeed, by specifying a user-name, a constant or a part of user-name (using BN = or SN =), the user constructs a dependency of the QS-database on the user-database. This dependency could become harmful because objects referenced in a query defined during a previous session can be removed from the user-database between the QS-session. Because of that, no answer

can be provided to such a query. Therefore, those objects database-keys cannot be recorded in the QS-database. The objects will be recorded in their character forms. An occurence of MATCH record-type is assigned to each match-string, i.e., name of an object contained in the user-database.

NUB

The NUB allows to record the hierarchy existing between sets and ranges. Indeed, a set can reference other sets or ranges within its definition. In the same way, it can be referenced by other sets. An occurence of NUB record-type is created each time a relationship between two sets exists.

Example :

If S1 is defined as S1 = S2 AND S3
and S4 as S4 = S3 OR S5 where Si are set-names

the relationship between these sets can be visualized in figure 4.5 .

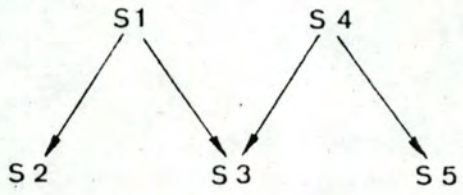


Figure 4.5

NUB occurences are created and linked to the sets. Figure 4.6 shows the created nubs. S1,S2,S3,S4 and S5 are occurences of SETOBJ record-type. S1 and S2 identify NUB1,S1 and S3 identify NUB2,... Specific access-paths link the sets to the nubs : S2 and S3 are referenced sets (access-path : REFD); S1 references other sets (access-path : REFS).

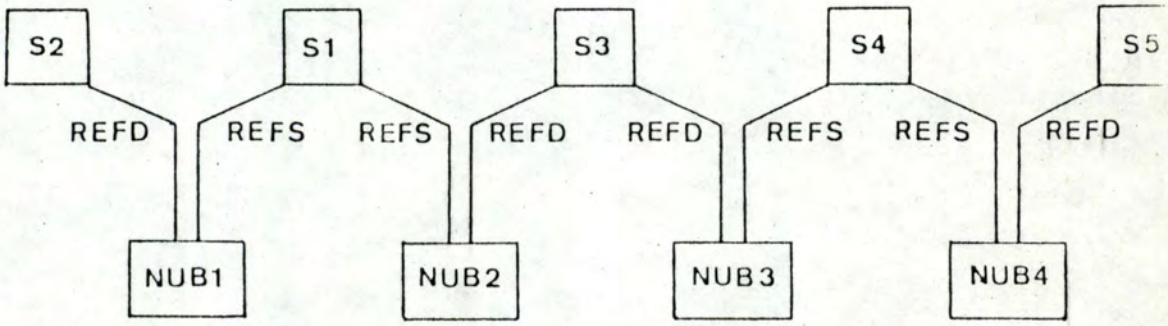


Figure 4.6

4.4.3. The Access-paths

ALLSET

It allows to access to the sets and ranges. ALLSET is sorted by alphabetic order of the item NAMSET.

ALLEXP

It allows from a set or a range to get its expression. ALLEXP is sorted by creation order.

ALLCOM

It allows from a set to get its comment lines describing this set. ALLCOM is sorted by creation order.

ALLCOM

It allows from a set to get its comment lines describing this set. ALLCOM is sorted by creation order.

ALLMAT

It allows, from a set or a range, to access to match-strings contained in their definitions. ALLMAT is sorted by creation order.

REFS

From a set, to get all the sets and ranges that the set references within its definition.

REFD

Allows from a set or a range to get all the objects that reference the given set or range

example

from figure 4.6, we start at S1 to find the sets it references. Using REFS, we find back NUB1 and NUB2; then, by using REFD, we find S2 and S3.

4.5. TWO ASPECTS OF THE QS REALIZATION

Two aspects of the Query System realization are discussed in this section. The first one concerns the strategy adopted in implementing the Query System and its consequences on the tests performed on it. The second one describes the conventions used in coding the QS program. Adopting these conventions will facilitate a further understanding of the program.

4.5.1. Strategy and Planning to Implement the QS

During the QS implementation, the coding strategy and the testing strategy are combined. We used the incremental implementation method at the main phases level : each of them was achieved independently of the others; afterwards, they were integrated two by two leading to the global program.

The order in which the phases were realized is as follows :

1. internal data structure
2. QS-database manager
3. synthesizer
4. parser
5. name selection
6. range builder
7. set and range manipulation

Adopting this strategy to implement the QS leads us to express the following criticisms.

Two facts can be derived from the reading of this list : the low level modules have been implemented first (phases (1) and (2)). As these two phases support the other ones, it was logical to implement them first. Secondly, the coding of the phases involved in the query evaluation (phases 3, 4 and 5) have not been achieved in the chronological order of application (see part 4.2). As these three phases are autonom ous, the implementation order didn't see to be of any importance once the interface between them is defined. Let us inspect these two aspects, based on our personal experience.

The choice to implement the low-level modules first has some consequences on the way these modules are tested. Indeed, this choice requires to build a driver-module to test these low-level modules. The input must be prepared within this module; this operation is time-consuming and tedious. Moreover, by testing these low-level subroutines, we get the impression that some of these tests are unuseful because we do not take into account the conditions in which those modules are going to be called. Finally, no concrete result can be shown to the user, namely, the person who expressed functional requirements. Therefore, the low-level modules are often unsufficiently tested.

The structure of the interface between the phases (3), (4) and (5) was defined before we coded these three phases. We found out that defining the output of a phase before really coding it is illusive because these outputs are still determinated at this low level. By designing the interface before all, we have to choose between drawing a unappropriate structure or performing feed-backs to modify phases already coded and tested. Therefore, we should propose to implement these phases in chronological order (namely, (4), (3) and then (5)). Moreover, by adopting this chronological strategy, we should benefit the sequence to get their outputs as inputs for the following phases. Thereby, the tests will be facilitated.

4.5.2. Conventions in Writing the Program

Conventions have been adopted to translate the algorithm into programming language. The purpose of these conventions is threefold. firstly, to facilitate the maintenance : by coding the subroutines according to the same structure, the maintener (often different from the designer in the ISDOS project) can easier draw the headlines of the module. Secondly, these conventions insure the program portability. Finally, their make easier the program development for the designer himself.

Instead of listing all the conventions adopted in the ISDOS project, we will draw the main ones from an example extracted from the QS program. The exhaustive list can be found in [WP267].

```
      SUBROUTINE DBCHCK (NAME,DBKEY,IRC)
C
C GIVEN THE NAME OF AN OBJECT, CHECK IF IT IS ALREADY
C DEFINED IN THE QS-DB
C
*CHAR,NAME(1)
      INTEGER DBKEY,IRC
C
C NAME  -> CHAR  NAME TO BE CHECKED
C DBKEY <- DBKEY DBKEY OF THE OBJECT  0 IF NONE
C IRC   <- RETURN CODE
C
*CHAR,NAMEF(30)
      INTEGER IRET
C.....
C
      CALL FFM (#ALLSET#,IRET)
C
C LOOP UNTIL FOUND OR EOF
C
   10 CONTINUE
         IF ( IRET == -1 ) GOTO 600
         IF ( IRET <> 0 ) GOTO 800
         CALL GFM (#NAMSET#,#ALLSET#,NAMEF,IRET)
         IF ( LSCOMP (NAME,1,NAMEF,1,30) ) GOTO 500
         CALL FNM (#ALLSET#,IRET)
         GOTO 10
C
C NAME FOUND GET KEY
C
   500 CONTINUE
       CALL (GKM (#ALLSET#,DBKEY,IRET)
       IF ( IRET <> 0 ) GOTO 800
       IRC = 0
       GOTO 900
C
C NAME NOT FOUND
C
   600 CONTINUE
       DBKEY = 0
       IRC = 0
       GOTO 900
C
C ERROR
C
    800 CONTINUE
        IRC = 1
        GOTO 900
C
C ALL DONE
C
    900 CONTINUE
        RETURN
        END
```

Figure 4.7

As illustrated in Figure 4.7, the following conventions have been respected :

- Coding in Extended Fortran :

  E.F. is a language designed in the ISDOS project [WP265]. As shown in the example, it includes the declaration of character variable storage (*CHAR), the substitution of relational and boolean operators (<> , & , == , ..), the substitution of named constants (#NAMSET#, #ALLSET#), ...

- The standardization of subroutines heading :

  It must contain a brief description of the subroutine purpose and the description of the input and output parameters.

- The standardization of the subroutines termination :

  One and only one "RETURN" is allowed in each subroutine; it is always preceded by the label "900".

- Incitement to introduce comment lines

- Labels :

  They must be ordered. Specific values are assigned to error cases.

CONCLUSIONS

The purpose of the thesis was to evaluate and improve the existing version of the Query System, a part of the SEM software developped under the ISDOS project. A new version is proposed and described. With regards to the previous one, the new version improves first the functionalities offered in the Query System : the possibilities for the queries expression have been extended with the introduction of the RANGE command and the underlying concept; functions geared towards facilitating the manipulation of sets and ranges have been doubled. These new features have been introduced to respond to needs expressed by the ISDOS software users. Finally, the implementation aspect was also reviewed and enhanced. By introducing two autonomous parsers and by isolating the internal data structure, the structure of the Query System is simplified and its maintenance is facilitated.

However, the proposed version could and should still be improved. Further realizations should concern first the implementation level : the selection of user-names on the basis of the criterion needs to be improved if we want to benefit from the new functions. Indeed, the user will hesitate to use the tool if it stays slow and hence desagreeable to use. From a functional view point, a mechanism to classify the queries is still required and should be implemented to complete the functions offered by the Query System.

This work was our first experience in software engineering covering the entire development life-cycle of a project from the requirements to the operation. Its framework was a large software engineering project with teams working in Michigan and in Namur. Hereunder, we present an evaluation of this first experience.

In developping the Query System, We have pointed out the following positive aspects:

- The project modularity is one of its major strong point. Its advantages are threefold : Firstly, it allows a fast understanding of the general software and facilitates the first approach of its structure. Secondly, one does not have to acquire a detailed knowledge of the entire project to be able to function effectively. Finally, the coding aspect is minimized : this is reduced to a sequence of calls to low-level subroutines.

- The use of the Extended Fortran Highly facilitates the software maintenance, improves the program legibility and ensures its portability.

- The conventions adopted within the ISDOS project also facilitate the maintenance and the program development for the designer himself.

- Last but not the least, the environment we benefit in the University of Michigan to develop our tool and the contrast with our own university shows how important it is to work in a good environment and points out the consequences of this factor on the results.

Some aspects could still be improved. We mentioned the conventions within the project as a main benefit of the ISDOS software. However, the design of the Query System shows the need for extending these conventions. They are justified by the number of persons working in the project and the relatively short time they spend on it. These conventions should propose a strategy to standardize the test procedure conducted in the project to improve the software reliability. Moreover, the need for a standardized documentation is essential : most of the time, the only document is the code itself. Therefore, conventions should require the designer to describe the main structure of his tool, the algorithms and the data structure. The computer scientists in general and in this software in particular have access to a tool to manage their documentations but they are reluctant to use them in their every day work.

The development of the Query System has shown us the preponderant role of the project manager, even if our student's status was particular. His function is to propose a schedule, to establish the work plan for the entire project , to coordinate the different persons involved and to distribute the ressources. However, such a framework shouldn't be too constraining on the designer because this could push him to rush through the design phases to produce intermediate results as fast as possible. This situation might be detrimental to the eventual end-product.

APPENDIX-1

The following appendix gives the definition of the example descibed in section I.1.5 in ISLDM language. Further informations about this language can be found in [WP279] .

1. OBJECTS

```
OBJECT PROCESS ;
     SYNONYMS PROC ;
     DOCUMENTATION ;
     Allows to describe a process, namely, an action performed by
     the system ;
     CODE NMPROC 30 ;


OBJECT MESSAGE ;
     SYNONYMS MSG ;
     DOCUMENTATION ;
     Allows to describe a message type interchanged between an
     INTERFACE and a PROCESS or between two PROCESS ;
     CODE NMMSG 31 ;;


OBJECT INTERFACE ;
     DOCUMENTATION ;
     Entity coming from the organization or from the environment
     the Information system interacts with by exchanging MESSAGES ;
     CODE NMCOND 32 ;
```

2. TEXTS

```
TEXT DESCRIPTION ;
     APPLIES ALL ;
     CODE CTDES 40 ;


TEXT PROCEDURE ;
     APPLIES PROCESS ;
     CODE CTPROC 41 ;
```

3. VALUE-RANGE

```
VALUE-RANGE ANYINTEGER ;
      VALUES NUMBER ;
```

4. PROPERTIES

```
PROPERTY PRIORITY ;
      APPLIES PROCESS ;
      DOCUMENTATION ;
      Define the priority level this process benefits. Name-constants a
      used for its value ;
      VALUES HIGH,LOW,MIDDLE,NONE ;
      CODE PPPRIO 101 ;


PROPERTY PERFORMING-TIME ;
      APPLIES PROCESS ;
      DOCUMENTATION ;
      This is the time needed to execute the described process. The tim
      is counted in mili-seconds ;
      VALUE INTEGER ;
      CODE PPPFTI 102 ;


PROPERTY PROBABILITY ;
      APPLIES CONDITION ;
      DOCUMENTATION ;
      This is the probability that the process will generate the messag
      VALUE REAL ;
      CODE PPPROB 103 ;
```

5. NAME-CONSTANTS

```
NAME-CONSTANT HIGH ;


NAME-CONSTANT LOW ;


NAME-CONSTANT MIDDLE ;


NAME-CONSTANT NONE ;
```

```
***** ON TERMINATION TRIGGERS *****


RELATION tgrs-term-rel ;
      PARTS triggor-part, triggered-part ;
      DOCUMENTATION ;
      The relationship express that on termination, a process
      can trigger another one ;
      COMBINATION triggor-part PROCESS WITH triggered-part PROCESS ;
      CONNECTIVITY MANY triggor-part, triggered-part ;
      CODE RTTGS 301 ;
      CONNECTION-TYPE S1 ;
      STORED triggor-part 1
             triggered-part 2 ;


STATEMENT tgrs-smt ;
      USED triggor-part, trgs-term-rel ;
      FORM  ON TERMINATION TRIGGERS triggered-part ;


STATEMENT tgred-smt ;
      USED triggered-part, trgs-term-rel ;
      FORM   TRIGGERED BY TERMINATION OF triggor-part ;


***** GENERATES IF *****


RELATION gnts-rel ;
      PARTS gnts-proc-part, gnts-msg-part, gnts-cond-part,
            gnts-number-part ;
      COMBINATION gnts-proc-part PROCESS
            WITH gnts-msg-part MESSAGE
            WITH gnts-cond-part CONDITION
            WITH gnts-number-part VALUE-FOR ANYINTEGER ;
      CONNECTIVITY MANY gnts-proc-part, gnts-msg-part
                  ONE gnts-cond-part, gnts-number-part ;
      CONNECTION-TYPE F3 ;
      CODE 302 ;
      STORED gnts-proc-part 3 ,
             gnts-msg-part 4 ,
             gnts-cond-part 1 ,
             gnts-number-part 2 ;


STATEMENT gnts-smt ;
      USED gnts-proc-part gnts-rel ;
      FORM  GENERATES gnts-number-part TIMES gnts-msg-part IF
            gnts-cond-part ;


STATEMENT gnted-smt ;
      USED gnts-msg-part gnts rel ;
      FORM  GENERATED gnts-number-part TIMES BY gnts proc-part IF
            gnts-cond-part ;
```

```
***** RECEIVES *****


RELATION rcvs-rel ;
      PARTS rcvs-msg-part, rcvs-receiver-part ;
      COMBINATION rcvs-msg-part MESSAGE
            WITH rcvs-receiver-part PROCESS, INTERFACE ;
      CONNECTIVITY MANY rcvs-msg-part, rcvs-receiver-part ;
      CONNECTION-TYPE S1 ;
      CODE RTRCVS 303 ;
      STORED rcvs-msg-part 1 ,
             rcvs-receiver-part 2 ;


STATEMENT rcvd-smt ;
      USED rcvs-msg-part rcvs-rel ;
      FORM RECEIVED BY rcvs-receiver-part ;


STATEMENT rcvs-smt ;
      USED rcvs-receiver-part rcvs-rel ;
      FORM RECEIVES rcvs-msg-part ;


***** GENERATES *****


RELATION gen-rel ;
      PARTS generts-part, genertd-part ;
      COMBINATION generts-part INTERFACE
            WITH genertd-part MESSAGE ;
      CONNECTIVITY MANY generts-part, genertd-part ;
      CONNECTION-TYPE S1 ;
      CODE RTGENE 304 ;
      STORED generts-part 1 ,
             genertd-part 2 ;


STATEMENT gens-smt ;
      USED genets-part, gen-rel ;
      FORM  GENERATES genetd-part ;


STATEMENT gend-smt ;
      USED genertd-part, gen-rel ;
      FORM GENERATED BY generts-part ;
```

APPENDIX - 2

Here follows the definition of the Query Language using the LANG-PAK meta-language. The definition of the elements contained in this meta-language can be found in table 3.3 .

```
-------------------------------
  High level query definition
-------------------------------

<query> = ( <univ-smt> ! <ren-smt> ! <comment-smt> ! <displ-smt> !
            <del-smt> ! <show-smt> ! <exec-smt> ! <hlp-smt> !
            <list-smt> ! <punch-smt> ! <read-smt> ! <range-smt> !
            <let-smt> ! <db-smt> ! <stop-smt> ) EOS ;


---------------------------
  SET command definition
---------------------------

<let-smt> = ( "LET"(220) ! "SET"(220) ) S( "=" V ) "=" <let-body> ;

<let-body> = ( "{"(3) <user-list> "}" ! <criterion> ) ;

<user-list> = ( <first-user-name> ) (0 9) <last-user-name> ;

<first-user-name> = S( "," ) "," ;

<last-user-name> = S( "}" V ) ;


-----------------------------
  RENAME command definition
-----------------------------

<ren-smt> = ("RENAME"(240) ! "RNM"(240) ) S("TO BE" V)
            ( "TO BE" ) (0 1) <set-name>   ;


-----------------------------
  Comment command definition
-----------------------------

<comment-smt> = ( "DELETE"(290) ! "DEL"(290) ! "ADD"(300) !
                  "REPLACE"(310) ! "REPL"(310) ) ( "COMMENT" ! "CMT")
                ("FROM" ! "FOR" ) (0 1) <set-name>  ;
```

Criterion definition

```
<criterion> = ( "("(1) ) (0 1) (<relational-part>) (0 99)
      ( "("(1) ) (0 1) (<not-sign>) (0 1) ( "("(1) ) (0 1)
      ( "["(19) <inv-in-rel> "]"(20) ! <m-type-of-query> )
      ( ")"(2) ) (0 3) ;

<relational-part> = ( "("(1) ) (0 1) ( <not-sign> ) (0 1)
      ( "("(1) ) (0 1) ( "["(19) <inv-in-rel> "]"(20) !
      <m-type-of-query> ) ( ")"(2) ) (0 2) <relational-operator> ;

<m-type-of-query> = ( "("(1) )(0 1) <type-of-query> ( ")"(2) ) (0 1)  ;


<type-of-query> = ("BN"(66) "=" <match-string> ! "SN"(67) "=" <match-string>
       "HAS SYNONYMS"(68) ! "DLC"(69) <operator>  <range-spec-date> !
       "MAXC"(70)  "=" I   ! "MINC"(71)  "=" I !  "TYPED"(72)       !
       "UNTYPED"(73)  ! "HAS TEXT"(74) !  "HAS PROPERTY"(75)        !
       "ISOLATED"(76) ! "HAS"(80) <textual-comment-name> ! <type-name> )
       ( ")"(2) ) (0 2) ;

<inv-in-rel> = S( "]" ) ;

<not-sign> = ( "NOT"(7) ! "~"(7) ) ;

<relational-operator> = ("AND"(5) ! "&"(5) ! "OR"(6) "!"(6) ) ;

<operator> = ("="(12) ! "<="(13) ! ">="(14) ! "<>"(15) ! "<"(10) !
        ">"(11) ) ;

<set-name> = <isdos-name> ;

<range-spec-date> = I ("/") (0 1) I ("/") (0 1) I  ;

<textual-comment-name> = S("(" V) (")"(2) ! <ppty-value> ) (0 1)  ;

<ppty-value> = ( N ! I ! S(V) ) ;

<statement> = S(V)  ;

<type-name> = S(")" V)  ;

<user-name> = <isdos-name>  ;

<match-string> = <isdos-name>  ;

<isdos-name> = S(":" "," "=" "!" "AND" "OR" "NOT" "(" ")" "{" "}" V) ;
```

```
-------------------------
 SHOW command definition
-------------------------

<show-smt> = ( "SHOW"(400) ) ("SET-NAMES"(62) ! "RANGES"(79) ! "RN"(78) !
      "RANGE-NAMES"(78) ! "SN"(62) ! "CRITERIA"(63) ! "CRTA"(63) !
      <set-name> )  ;


----------------------------
 DISPLAY command definition
----------------------------

<displ-smt> = ( "DISPLAY"(200) ! "DISPL"(200) ) ("SET-NAMES"(62) !
       "UNIVERSE"(77) ! "SN"(62) ! "CRITERIA"(63) ! "CRTA"(63)   !
       "UNIV"(77) ! "RN"(78) ! "RANGES"(79) ! "RANGE-NAMES"(78)  !
       <set-name> )  ;


-------------------------------------------------
 DESTROY, UNDESTROY and ERASE commands definition
-------------------------------------------------

<del-smt> = ( "DESTROY"(320) ! "UNDESTROY"(380) ! "UNDES"(380) !
        "ERASE"(210) ) <set-list>  ;


----------------------------
 EXECUTE command definition
----------------------------

<exec-smt> = ("EXECUTE"(270) ! "("EXEC"(270) ) ("FROM") (0 1) (S) (0 1) ;


-------------------------
 LIST command definition
-------------------------

<list-smt> = "LIST"(230)  ( S"( "=" V ) ! S ) ("=" <let-body> !
        <let-body> ) (0 1) (<arg>) (0 1)   ;

<arg> = ("PRESENTATION"(58) ! PRSNT"(58) ) ("BASIC"(64) !
        "SYNONYM(65)  )  ;
```

```
-----------------------------------
  SET UNIVERSE command definition
-----------------------------------

<univ-smt> = "SET" ("UNIVERSE"(250) ! "UNIV"(250) ) "="
          ("DB"(52) ! <let-body> )  ;


---------------------------
  READ command definition
---------------------------

<read-smt> = "READ"(280) <set-name> ("FROM"(777) ) (0 1) (S) (0 1) :


-----------------------------
  PUNCH statement definition
-----------------------------

<punch-smt> = "PUNCH"(360) (<first-set-name>) (0 9) <last-set-name>
          ( "ON"(777) S("NOEMPTY" "EMPTY" V ) ) (0 1) ("EMPTY"(59) !
          "NOEMPTY"(60) ) (0 1)   ;

<first-set-name> = S(",")  ","  ;

<last-set-name> = S(V)  ;


---------------------------
  RANGE command definition
---------------------------

<range-smt> = "RANGE"(410)  <range-name> "=" ( <list-value-ra> !
       <compare> ! <compare-1> ! <two-borders> )  ;

<list-value-ra> = "{"(8) <list-r>  "}"  ;

<compare> = ("LOWER-THAN"(10) ! "LT"(10) ! "GREATER-THAN"(11) ! "GT"(11) !
        "LOWER-EQUAL"(13) ! "LE"(13) ! "GREATER-EQUAL"(14) ! "GE"(14) !
        "EQUAL"(12) ! "EQ"(12) ! "NON-EQUAL"(15) ! "NE"(15) ) ( N ! I !
        "'"(122) S("'") "'"  )  ;

<compare-1> = ("EQUAL"(12) ! "EQ"(12) ! "NON-EQUAL"(15) ! "NE"(15) ) S ;

<two-borders> = ( N ! I ! "'"(122) S("'") "'" ) "THRU" ( N ! I ! "'"(122)
        S("'") "'" )  ;

<list-r> = (<value-r>) (0 9) <last-value-r>  ;

<value-r> = ( N ! "'"(122) S("'") "'" ! S(",") ) ","  ;

<last-value-r> = ( N ! "'"(122) S("'") "'" ! S("}" V) )  ;

<range-name> = S("=" V)  ;
```

```
-----------------------------------------
  SAVE and RESTORE commands definition
-----------------------------------------

<db-smt> = ("RESTORE"(370) ! "REST"(370) ! "SAVE"(330) ) ("ALL"(51) !
        <set-list> )  ;

<set-list> = ( <first-set-name> ) (0 9) <last-set-name> ("RELATED"(54))
        (0 1)   ;


                ---------------------------
                  STOP command definition
                ---------------------------


<stop-smt> = "STOP"(260)  ;


                ---------------------------
                  HELP command definition
                ---------------------------


<hlp-snt> = "HELP"(350) (<command>) (0 1)  ;

<command> = ( "DISPLAY"(200) ! "DSPL"(200) ! "ERASE"(210) ! "LET"(220) !
       "LIST"(230) ! "RENAME"(240) ! "RNM"(240) ! "SET UNIVERSE"(250) !
       "SET UNIV"(250) ! "RANGE"(410) ! "STOP"(260) ! "DELETE COMMENT"(290) !
       "DEL COMMENT"(290) ! "DEL CMT"(290) ! "DELETE CMT"(290) !
       "ADD COMMENT"(300) ! "ADD CMT"(300) ! "REPLACE COMMENT"(310) !
       "REPLACE CMT"(310) ! "REPL COMMENT"(310) ! "REPL CMT"(310) !
       "EXECUTE"(270) ! "EXEC"(270) ! <com-next> )  ;

<com-next> = ( "READ"(280) ! "PUNCH"(360) ! "SAVE"(330) ! "RESTORE"(370) !
       "REST"(370) ! "DESTROY"(320) ! "SET"(220) ! "SHOW"(400) ) ;


                ---------------------------
                  READ command definition
                ---------------------------


<read-smt> = "READ"(280) <set-name> ("FROM"(777) ) (0 1) (S) (0 1) :
```

APPENDIX-3

Here follows the description of the QS-database expressed in the DDL of ADBMS. Further informations on ADBMS can be found in [WP191].

Records

```
RECORD SETOBJ
ITEM   NAMESET   CHAR    30
ITEM   DELSET    INTEG   16
ITEM   TYPSET    INTEG   16

RECORD EXPRES
ITEM   LINEXP    CHAR    80

RECORD COMMEN
ITEM   LINCOM    CHAR    80

RECORD PARSED
ITEM   LINPAR    INTEG   16

RECORD MATCH
ITEM   MATPAR    CHAR    80

RECORD NUB
```

Sets

```
SET     ALLSET  SORTED    NAMSET
OWNER   SYSTEM
MEMBER  SETOBJ


SET     ALLEXP  FIFO
OWNER   SETOBJ
MEMBER  EXPRES


SET     ALLCOM  FIFO
OWNER   SETOBJ
MEMBER  COMMEN


SET     ALLPAR FIFO
OWNER   SETOBJ
MEMBER  PARSED


SET     REFD    FIFO
OWNER   SETOBJ
MEMBER  NUB


SET     REFS    FIFO
OWNER   SETOBJ
MEMBER  NUB


SET     ALLMAT FIFO
OWNER   SETOBJ
MEMBER  MATCH
```

REFERENCES

[ASTR.75] M.M. Astrahan, D.D. Chamberlin
"Implementation of a Structured English Query Language "
Communications of the ACM , Vol 18 Nr 10, Oct 1975.

[ATZE.81] P. Atzeni and P.P. Chen
"Completeness of query languages for the Entity-Relationship model "
Entity-Relationship Approach to Information Modeling and Analysis, E.R. Institute, 1981.

[BODA.83] F. Bodart, Y. Pigneur
"Conception Assistee des Applications Informatiques. Premiere Partie : l'etude d' Opportunite et l' Analyse Conceptuelle",
Ed. Masson, Paris. Expected.

[BUNE.82] P.Buneman, R.E. Frankel, R. Nikhil,
"An Implementation Technique for Database Query Languages ",
ACM Transactions on Database Systems, Vol 7, Nr 2,June 1982.

[CHAM.76] D.D.Chamberlin and all
"SEQUEL 2 : A unified Approach to Data Definition, Manipulation and Control ".
IBM research Dvlpt 20, Nov 1976, pp 560-575.

[CHEN.76] P.P. Chen
"The Entity-Relationship model - Toward a unified view of data",
ACM Transactions on Database Systems, Vol 1, Nr 1, March 1976.

[DATE.77] C.J. Date
"An Introduction to Database Systems" (2nd Ed.),
Addison-Wesley Reading, Mass., 1977.

[DSL.82] "DSL : manuel de references"
Institut d'Informatique, Facultes de Namur, 1982.

[GHOS.77] S.P. Ghosh
"Data Base Organization for Data Management",
Academic Press, New York, 1977.

[HEIN.75] L.E. Heindel, J.T. Roberto
"LANG-PAK, An Interactive Language Design System"
Elsevier Publishing Company, Inc. New York, 1975.

[HERO.80] C.F.Herot,
"Spacial Management of Data",
ACM Transactions on Database Systems, Vol 5, Nr 4, Dec 1980,
pp 493-514.

[HUTT.79] A.T.F. Hutt,
"A Relational Database Management System",
Wiley and sons, Hampshire, 1979.

[IM47] B. Geubelle,
"A Query System for the System Encyclopedia Manager",
ISDOS Internal Memorandum Nr 47, 1982.

[KIM.82] W. Kim,
"On Optimizing an SQL-like Nested Query",
ACM Transactions on Database Systems, Vol 7, Nr 3, Sept
1982.

[PIRO.74] A. Pirotte, P. Wodon,
"A Comprehensive Formal Query Language for the Relational
Database : FQL"
MBLE Research Laboratory Report R283, Dec 1974.

[PIRO.75] A.Pirotte
"Comparaison de langages d'interrogation de bases de donnees
relationnelles"
Technical note N100, Ecole d'Ete de l' AFCET, Rabat, July
1975.

[PSL.82] "PSL / PSA user's reference Manual version 5.2",
nr ref 171, ISDOS project, University of Michigan, July 82.

[REIS.81] P. Reisner
"Human Factor Studies of Database Query Languages : a Survey and Assessement",
ACM Computing Surveys, Vol 13, Nr 1, March 1981.

[ROSE.80] D.J. Rosenkrantz, H.B. Hunt
"Processing Conjunctive Predicates and Queries",
Proceedings on Very Large Data Bases, 6th conference, Oct 1980.

[STEM.78] D.W. Stemple, C. Welty, M. Becker, W.Mayfield,
"Tablet : The Algebra Based Language For Enquiring of Tables",
Technical Report 79/19, Computer and Information Science Dept, Univ Massachussetts, Nov 1978.

[STON.76] M. Stonebraker, E. Wong , P. Kreps, G. Held
"The Design and Implementation of INGRES",
ACM Transactions on Database Systems, Vol 1, Nr 3, Sept 1976.

[TEICH.79] D. Teichroew, P. Macasovic, E.A. Hersley, Y. Yamamoto,
"Application of the Entity-Relationship Approach to Information Processing Systems Modelling"
Proceedings, International Conference on Entity-Relationship Approach to System Analyzis and Design, Dec 10-12, 1979, Los Angeles, pp 23-51.

[TM119] D. Marcellus, K.C. Kang
"A Query System for Generalizied Analyzer G1.0",
ISDOS Technical Memorandum 119, Sept 1978.

[TM427] K.C. Kang
"Suggested improvements of the Query System",
ISDOS Technical Memorandum 427, April 1982.

[WELT.81] C. Welty, D. Stemple
"Human Factor Comparison of a Procedural and a Non-procedural Query Language"
ACM Transactions on Database Systems, Vol 6, Nr 4, Dec 1981.

[WP191] "A Database Management System (ADBMS) based upon DBTG 71",
ISDOS Working Paper Nr 191, ISDOS Project, Ann Arbor, MI, Feb 1981.

[WP265] E.A. Hershey III, Y. Yamamoto, E. Chikosky, D. Mielta,
"Extended FORTRAN (EF)",
ISDOS Working Paper Nr 265, ISDOS Project, Ann Arbor, MI, July 1979.

[WP267] E.A. Hershey III, Y. Yamamoto, E. Chikosky, D. Mielta,
"ISDOS Software Conventions",
ISDOS Working Paper Nr 267, ISDOS Project, Ann Arbor, MI, July 1979.

[WP279] "Information System Language Definition System Language User's Manual Version M1.2",
ISDOS Working paper Nr 279, ISDOS Project, Ann Arbor, MI, Feb 1980.

[WP456] "System Encyclopedia Manager (SEM) Query System User's Manual",
ISDOS Working paper Nr 456, ISDOS Project, Ann Arbor, MI, Jan 1983.

[YAO.79] S.B. Yao
"Optimization of Query Evaluation Algorithms",
ACM Transactions on Database Systems, Vol 4, Nr 2, June 1979.

[YOUR.79] E. Yourdon, L.L. Constantine,
"Structured Design",
Prentice-Hall, NewJersey, 1979.

[YUZO.81] Y. Yamamoto
"An Approach to the Generalization of Software Life Cycle Support Systems",
Ph.D. Thesis, The University of Michigan, 1981.

[ZLOO.80] M.M. Zloof,
"Query by Example",
ACM Computing Surveys, Vol 13, Nr 1, March 1980.